高等院校计算机类课程“十二五”规划教材

C#程序设计

C# CHENGXU SHEJI

主　编◎陈　锐　李　佳

副主编◎李绍华　方　洁　雷　军

合肥工业大学出版社

高等学校“十一五”省级规划教材

C 语言程序设计

（第 4 版）

主　编　吴国凤

副主编　黄存东　梅灿华　黄　兵

编　委　（按姓氏笔画排序）

王　胜　李中永　李家兵

吴国凤　范文广　孟林树

宣善立　贺继东　黄存东

黄　兵　梅灿华

主　审　王　浩

合肥工业大学出版社

内 容 简 介

本书是面向高职高专学生学习C语言程序设计的理想教材。全书共分12章，主要内容包括：C语言概述、数据类型与基本运算、顺序结构、选择结构、循环结构、数组与字符数据处理、函数、指针、结构体与联合、位运算、文件等。

本书内容丰富，重点突出，针对初学者的特点，讲解简明扼要，深入浅出，通俗易懂。本书对C语言的语法规则进行了提炼，注重讲解程序设计的概念和方法，培养学生编写程序的能力。各章配备丰富的例题、习题，并对典型例题进行精解，另配《C语言程序设计实训教程》一书，特别适合作为工科高等院校各专业程序设计语言和高职高专各专业的课程教材，也可作为自学教材。

图书在版编目(CIP)数据

C语言程序设计/吴国凤主编. —4版. —合肥：合肥工业大学出版社，2012.8

ISBN 978-7-5650-0840-5

Ⅰ.①C… Ⅱ.①吴… Ⅲ.①C语言—程序设计—高等学校—教材 Ⅳ.①TP312

中国版本图书馆CIP数据核字(2012)第180582号

C语言程序设计(第4版)

主编 吴国凤 **责任编辑** 陆向军

出 版	合肥工业大学出版社	**版 次**	2012年8月第4版
地 址	合肥市屯溪路193号	**印 次**	2012年8月第7次印刷
邮 编	230009	**开 本**	787毫米×1092毫米 1/16
电 话	综合编辑室：0551-2903028	**印 张**	16
	发 行 部：0551-2903198	**字 数**	360千字
网 址	www.hfutpress.com.cn	**发 行**	全国新华书店
E-mail	hfutpress@163.com	**印 刷**	安徽江淮印务有限责任公司

ISBN 978-7-5650-0840-5 定价：26.00元

计算机系列教材编委会

第4版前言

《C语言程序设计》自2005年8月初版以来，承蒙学术界同行和广大读者的厚爱，纷纷采用本书作为高职高专计算机类专业教材，使本书的发行量迅速增加。2007年10月，本书被列为安徽省“十一五”规划教材。虽然如此，实践表明本书仍存在不足之处。为了保证本书的先进性和实用性，我们进行了多次修订。

本次修订主要是调整部分章节内容和图表，增加了课后练习，力求使本书趋于完美。

C语言是目前国内外广泛使用的程序语言之一。C语言功能丰富、表达能力强、使用灵活方便、程序执行效率高、可移植性好。C语言的模块化、结构化特性使其可以满足现代程序设计的需要，既可以用来编写系统软件，又可以用来编写应用软件。因而，C语言是软件工作者必须掌握的一个工具。

为了满足高职高专教学改革的需要，我们组织了省内有关高校长期在教学第一线的、有经验的教师编写了《C语言程序设计》一书，供高职高专的师生们使用。为适合教学方式的变革，我们在编写教材的同时，一并编写了配套的实验教材、课件、电子教案及相应的程序设计素材库。

本书共分12章，全面、系统地讲述了C语言的基本概念、原理和方法。教材组织精练，讲解深入浅出，循序渐进。特别强调理论联系实际，通过大量的案例解析，说明了语法结构、编写程序的方法以及程序设计的技巧。本书既注重培养学习者程序设计的能力，又提倡良好的程序设计风格。另外，每章还配备了大量的习题，便于学生练习掌握。本书由吴国凤担任主编，黄存东、梅灿华、黄兵担任副主编。全书由吴国凤统编定稿。

虽然经我们多次认真的修订、补充和校正，但由于我们的理论水平、研究能力和知识深广度的限制，书中难免还存在缺点和错误。真诚希望同行专家和广大读者批评指正。

编　者

2012年8月

目　录

第 1 章　C 语言概述

【教学提示】

本章主要介绍 C 语言及其程序设计的基本知识。通过案例分析，了解 C 语言的简史和特点；掌握 C 语言程序基本结构；掌握计算机算法的基本概念；掌握 C 程序的运行环境与学习方法。

【核心概念】

C 程序的构成　main 函数和其他函数　C 程序的书写风格　计算机算法　C 程序的运行环境

1.1　程序设计与 C 语言

1.1.1　程序与程序设计语言

1. 程序

计算机最能吸引人之处是它能自动执行指定的程序。所谓程序，就是用语言、文字、图表等方式表达解决某个问题的方法和步骤，是计算机解决某些特定问题所需的代码化指令序列。程序设计者根据预先制定的功能和规则，编写一系列完整指令，由计算机执行，实现预定的功能和任务，这就是计算机程序。

2. 程序设计语言

程序设计就是根据给定的任务，设计、编写、调试程序的过程。在程序设计过程中要用一定的工具以及环境，C 语言就是一种常用的计算机高级程序设计语言工具。程序设计语言的种类很多，大体上分为 3 大类。

(1)机器语言

机器语言是用 0 和 1 表示的二进制代码的指令序列，是计算机能直接识别和执行的语言。机器语言中的每一条语句(称为指令)由操作码和操作数组成，反映了操作的对象和性质。但由于机器语言是用二进制表示，难于学习和记忆，编写工作量大，程序可读性差，非计算机专业人员难于掌握。

(2)汇编语言

汇编语言采用助记符号表示机器语言中的指令和数据，即用助记符号代替了二进制表示的机器指令，例如用 ADD(Addition)表示做加法的助记符，每条汇编语言的指令对应一条机器语言的指令。

计算机硬件只能识别机器指令。要执行汇编语言编写的源程序，必须用汇编处理程序将源程序翻译成机器语言程序才能执行。

(3)高级语言

高级语言表达方式接近被描述的问题，接近于自然语言和数学表达式，易于人们接受和掌握。随着计算机技术的发展，形成了为数众多的计算机高级语言并得到广泛应用，每种高

级语言都有其适合的应用领域。

如:FORTRAN 语言适用于数值计算;BASIC 语言简单易学,适合于初学者学习;Pascal 语言适用于教学;C 语言适用于系统软件和应用软件的开发;Java 语言既是通用的程序设计语言,又是可以用于 Internet 开发的面向对象的程序设计语言。

由于计算机只能识别机器语言,因此,用高级语言编写的程序(或称源程序)要通过专用的程序将高级语言翻译成机器语言程序(或称目标程序)后才能被计算机执行。将高级语言翻译成机器语言的方式有两种:一种是将高级语言程序一次性翻译成机器语言,再通过连接程序将系统库连接到程序中形成可执行程序,然后运行,这种语言称为编译型语言,如图 1-1 所示;另一种是翻译一句执行一句,称为解释型语言。

图 1-1 编译型语言的执行过程

C 语言是典型的编译型语言。面向过程的计算机的语言的核心是将输入数据通过算法转换为输出数据。因此,程序主要描述的是:数据的表示,对数据的处理算法以及数据输入输出。

1.1.2 C 语言的发展与特点

1. C 语言的发展

C 语言是在 B 语言的基础上发展起来的。它的前身是 ALGOL 60(1960 年,面向问题的高级语言,离硬件比较远,不宜用来编写系统程序)。1963 剑桥大学推出了 CPL(复合程序设计语言),CPL 语言接近硬件一些,但规模较大,难以实现。1967 年剑桥大学的马丁·理查德对 CPL 进行了简化,推出了 BCPL(基本复合程序设计语言)。1970 年美国贝尔实验室的肯·汤普逊对 BCPL 进行了进一步的简化,突出敢硬件处理能力,并取了“BCPL”的第一个字母“B”作为新语言的名称,并且用 B 语言编写了第一个 UNIX 操作系统。但“B”语言过于简单,功能有限,在 1972 年贝尔实验室的布朗.W. 卡尼汉和丹尼斯.M. 利奇对 B 语言进行了完善和扩充,在 B 语言的基础上设计出了 C 语言。C 语言保持了 BCPL 语言和 B 语言精练、接近硬件的优点,又克服了它们过于简单、数据无类型、功能有限的缺点。后来多次改进,在贝尔实验室内部使用。直到 1975 年 UNIX 第 6 版公布突出优点才引起人们普遍注意。

1977 年,为了让 C 语言脱离 UNIX 操作系统,成为在任何计算机上都能运行的通用计算机语言,布朗.W. 卡尼汉和丹尼斯.M. 利奇(K&R)合著了著名的《C 程序设计语言》一书,对 C 语言的语法进行了规范化的描述,成为当时的标准。通常简称为《K&R》标准。但是,在《K&R》中并没有定义一个完整的标准 C 语言,1983 年美国国家标准委员会对 C 语言进行了标准化,颁布了第一个 C 语言标准草案为 83 ANSI C;1987 年又颁布了新标准为 87 ANSI C;1990 年国际标准化组织 ISO(International Standard Organization)接受 87 ANSI C 为 ISO 的标准。目前流行的 C 编译系统都以此为基础。

2. C 语言的特点

C 语言同时具有汇编语言和高级语言的优势,它把高级语言面向过程和低级语言与硬

件关系密切的优点有机地结合起来，具有许多显著的特点：

(1)语言简洁、紧凑，使用方便、灵活。ANSI C 为标准只有 32 个关键字(详细见第 2 章 2.1.1)；9 种控制语句；数据构造能力强；程序书写格式自由。与 Pascal 相比，源程序短，压缩了一切不必要的成分，见表 1-1 所列。

表 1-1　C 语言与 Pascal 语言比较

C 语言	Pascal 语言	含义
{ }	BEGIN.... END	复合语句
if(e) s;	IF(e)　THEN s	条件语句
int i;	VAR i:INTEGER	定义 i 为整型变量
int a[10];	VAR a:ARRAY[1..10] OF INTEGER	定义整型数组 a
int f();	FUNCTION f():INTEGER	定义 f 为返值整型的函数
int * p;	VAR p:↑INTEGER	定义 p 为指向整型变量的指针
i+=2;	i=i+2	赋值 i+2⇒i
i++,++i;	i=i+1	i 自增值

(2)运算符和数据结构类型丰富。C 语言有 34 种运算符(见附录 2)。并把括号、赋值、逗号等都作为运算符处理。从而使 C 的运算类型极为丰富，可以实现其他高级语言难以实现的运算。

C 语言的数据类型有：基本类型、构造类型(数组、结构体、联合体)、指针类型和空类型，为处理各种复杂数据类型提供了实用的手段。

(3)具有结构化的控制语句。C 语言具有多种结构化的控制语句(if～else、for、while、do～while 等)，可以很容易地实现结构化的各种基本结构，用来设计结构化程序。

(4)功能强大。C 语言既可用于书写系统软件，也可用于书写应用程序。

(5)生成的目标代码质量高，程序执行效率高。

(6)C 语言允许直接访问物理地址，具有一定的低级语言的特性。

(7)移植性好。在某种计算机系统中的 C 程序，只要少许改动，甚至不作改动就可用于各种型号的计算机和各种操作系统。

1.2　C 程序初识

1.2.1　简单 C 程序案例

为了让初学者对 C 语言程序有初步的认识，下面通过几个简单的案例，介绍 C 程序设计语言的特点以及 C 源程序的基本构成。

【例 1.1】　最简单的 C 语言程序，在屏幕上输出“This is a C Program.”。

```
#include <stdio.h>
void main(void)
```

```
{
    printf("This is a C Program.\n");
}
```

程序运行结果,如图 1-2 所示。

图 1-2 例 1.1 程序运行结果

【解析】

其中,第 1 行 #include <stdio.h>为文件包含,基本输入/输出函数原型及有关信息包含在"stdio.h"库文件中(或称头文件),有关文件包含命令的使用,请参阅第 9 章;

第 2 行 main 被称作"主函数",在任何一个 C 程序中都必须有一个且只有一个 main 函数。在 main()函数中有很多 C 语句,用一对大括号括起来,称为函数体。

第 3 行左大括号"{"代表开始,最后一行右大括号"}"代表结束;

在本例的程序中,函数体中只包括一条语句,printf()是 C 语言中的输出函数,作用是将双引号之间的内容原样输出到屏幕上。"\n"是换行符,即在输出"This is a C Program."后回车换行。

【例 1.2】 求两个整数 a、b 累加和。

```
#include <stdio.h>
void main(void)
{
    int a,b,sum;                    /*定义变量*/
    a=10;b=8;                       /*给变量赋值*/
    sum=a+b;                        /*求累加和*/
    printf("%d+%d=%d\n",a,b,sum);   /*输出结果*/
}
```


程序运行结果,如图 1-3 所示。

图 1-3 例 1.2 程序运行结果

【解析】程序中,第 1 行、第 2 行、第 3 行含义同上;第 4 行为变量说明语句,说明本程序需用 3 个整型变量;第 5 行是赋值语句,表明 a、b 变量的初值分别为 10、8;第 6 行将 a+b 的累加和送入变量 sum 中;第 7 行是将运算的结果输出到屏幕上。printf()中的"%d"为格式说明符,表示在这个位置上将用一个整型数值代替。

"/*…*/"表示注释,注释部分不参与也不影响程序的运行,这些注释只是用来帮助人们阅读和理解程序,注释内容可以加在程序的任何位置。

C 程序中的语句最重要的一个特点就是每条基本语句的后面都要有一个分号,作为语句的结束标志。

【例 1.3】 输入两个整数,输出两个数中较大的数。

```
#include <stdio.h>
void main(void)
{
    int x,y,imax;
```

```
    scanf("%d,%d",&x,&y);
    if (x>y) imax=x;
    else imax=y;
    printf("\nThe Max number is %d\n",imax);
}
```

若从键盘输入:5,8 ↙,则程序运行结果,如图 1-4 所示。

图 1-4　例 1.3 程序运行结果

【解析】此程序在执行时,先由 scanf()函数从键盘读取两个数据,这时由用户从键盘上输入 5,8 ↙(↙表示回车键),表示将 2 个整型值,分别存放到 x 和 y 存储单元。if 语句用来比较 x 和 y 值的大小,取其中大数赋给最大值变量 imax。最后通过 printf 函数将大数输出到屏幕上。

C 语言实际是一个函数式的语言,可以把一个复杂程序分解成若干个函数(模块)来完成,每个函数来实现特定的功能。对上一题,我们也可以用主函数和子函数合作的形式来完成。

【例 1.4】 输入两个整数,输出两个数中较大的数(用子函数的方法)。

```
#include <stdio.h>
int Max(int, int);                    /* 子函数原型声明 */
void main(void)
{
    int x,y,imax;
    scanf("%d,%d",&x,&y);             /* 任意输入两个数 */
    imax=Max(x,y);
    printf("\nThe Max number is: %d\n",imax);
}
int Max(int x,int y)
{   int z;
    if (x>y) z=x;
    else z=y;
    return(z);
}
```

若从键盘输入 20,12 ↙,程序运行结果,如图 1-5 所示。

图 1-5　例 1.4 程序运行结果

【解析】此程序包括两个函数,一个是主函数 main(),另一个子函数 Max()用于求两个数 x、y 中较大数。在执行时,先由 scanf()函数从键盘读取两个数据,这时由用户从键盘上输入 20,12 ↙(↙表示回车键)。此时 x 被赋值 20,y 被赋值

12。然后执行函数调用语句:imax＝Max(x,y),将 x、y 的值传入 Max 函数中。在 Max 函数中经过判断后,z 中的值就是两个数的较大值。用 return 语句将 z 的值返回函数调用处,然后继续执行主函数,直到程序结束。

1.2.2 C 程序基本构成

通过上述例子可以看出,C 语言程序结构大体上由下面几个部分组成。

1. 程序的组成

每个程序一般由程序主体、编译预处理和注释三部分组成。

(1)程序主体:函数定义是 C 程序的主体部分,程序的功能由函数来完成。

(2)编译预处理:每个以符号“#”开头的行,称为编译预处理行,是 C 提供的一种模块工具。

(3)注释:是用“/* …*/”括起的内容,其作用给程序设计者一种提示或记号。注释内容不参加程序的执行,主要是为了提高程序的可读性。

2. 函数的组成

(1)每个函数(包括主函数)的定义分为两个部分:函数首部和函数体。

(2)函数首部包括:返回值的类型,函数名,形式参数表。

函数首部的格式:

返回值的类型 函数名(type 形参 1,type 形参 2,….)

例如:int Max(int x,int y)

(3)函数体是由一对{}括起的语句序列,其中包括:变量说明部分和实现函数功能的语句两部分组成。

函数体的格式:

```
{
    变量说明部分
    实现函数功能的语句
}
```

例如:

```
int Max(int x,int y)          /* 函数的首部 */
{                             /* 代表函数体开始 */
    int z;                    /* 变量说明 */
    if (x>y) z=x;             /* 具体实现函数功能的语句 */
    else z=y;
    return(z);
}
```

3. 语句的组成

(1)语句是组成程序的基本单元。每个语句最后要用分号(;)结尾,分号是语句的组成部分。但要注意,函数说明语句后面不能有分号。如上例中 void main(void)和 int Max(int x,int y)后面不能加分号。

(2)C 程序中的语句含有各种关键字、运算符、表达式等。例如:int 为关键字,用来说明

变量的类型；x＞y 中的“＞”为关系运算符；z＝x 为赋值表达式。

(3)C 程序的书写格式比较自由，可以在一行上写若干语句，也可以将一条语句写在多行上。

注意：在 C 语言程序中英文字母是区分大、小写的。在定义变量时，同样的字母的大、小写代表不同的变量，而 C 语言的关键字和基本语句都是用小写字母表示。

4. 程序结构特点

(1)C 程序是由一个或多个函数构成。每个 C 程序有且仅有一个主函数，函数名规定为 main()函数，它由系统指定的。除主函数外，可以有一个或若干个子函数(如 Max 函数)，并且在主函数中调用它。

(2)主函数是整个程序的主控模块，是程序的入口，C 程序总是从 main()函数开始执行，最终在 main()函数中结束。main()函数可以放在程序的任意位置。

(3)函数在执行过程中可以根据程序的需要调用系统提供的库函数，例如在程序用到的 scanf()/printf()函数，它们的原型包含在 stdio. h 文件中。因而在今后的应用案例中，若要调用系统提供的库函数，则在调用前必须将被包含文件嵌入到源文件中。

(4)C 语言的变量在使用之前必须先定义其数据类型，未经定义的变量不能使用。定义变量类型的语句应在可执行语句的前面，如上例中 main()函数中的第一个语句就是变量定义语句，它必须放在第一个执行语句 scanf()的前面。

(5)C 语言中提供了丰富的函数，被称作库函数。标准 C 中提供了 100 多条库函数，Turbo C 和 MS C 中提供了 300 多条库函数。利用这些系统提供的函数可以非常轻松地编写一些功能强大的程序。

(6)主函数可以调用任何其他函数；任何非主函数都可以相互调用，但不能调用主函数。

重点：函数是程序的基本单位；语句要以分号作为结束标志。

1.3　程序设计方法与算法

1.3.1　程序设计方法

从 1.2 节中，我们已经初步了解了 C 语言程序设计的过程以及程序的基本组成。但在程序设计中需要用一定的方法来指导，以便提高程序的可读性、维护性、稳定性及编程的效率。目前有两种重要的程序设计方法：结构化程序设计方法和面向对象程序设计方法。

1. 结构化程序设计方法

结构化程序设计(Structured Programming)方法是由 E. Dijkstra 等人于 1972 提提出来的，它建立在 Bohm、Jacopini 证明的结构定理的基础上。结构定理指出：任何程序逻辑都可以用顺序、选择和循环三种基本结构来表示。实验证明：结构化程序设计方法确实使程序执行效率提高，并且由于减少程序的出错率，从而大大减少了维护的费用。

结构化程序设计有两个主要特点：

(1)程序总是由三种基本结构组成，即顺序结构、选择结构和循环结构。三种控制结构

共有的特点:单入口、单出口;无死语句;无死循环。其流程框图如图1-6所示。

(2)自顶向下,逐步求精和模块化是结构化程序设计方法中最典型、最具有代表性的方法。

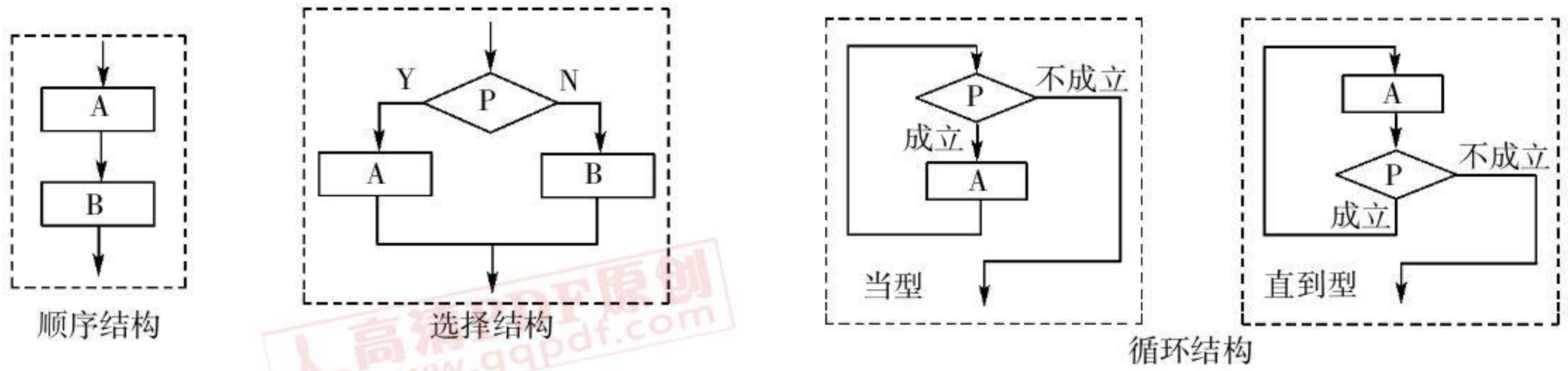


图1-6 三种控制结构

2. 面向对象程序设计方法

面向对象的程序设计是另一种重要的程序设计方法。在面向对象的程序设计方法中,一个程序由若干个对象组成,对象之间通过消息相互作用。

面向对象程序设计语言主要有三个特征:封装性、继承性和多态性。

(1)封装性:对象是一个封装体,在其中封装了该对象所具有的属性和操作,对象作为操作的基本单元,实现了数据和数据处理相结合。一般用类来实现封装。

(2)继承性:一种支持重用的思想,用现有的类型派生出新的子类。

(3)多态性:是指在类中定义的属性或行为被特殊继承后,可以具有不同的数据类型或表现出不同的行为。

C语言是一个面向过程的语言,它所采用的是用结构化程序设计方法;C++是一个面向对象的语言,因而采用的面向对象程序设计方法。C是C++的基础,C++是C的超集。当我们掌握了C语言,再去学习C++,达到事半功倍的效果。

重点:自顶向下,逐步求精和模块化是结构化程序设计中最典型的方法。

1.3.2 算法

算法是解决问题的逻辑步骤,是对特定问题求解步骤的一种描述。只有通过算法能够表示的的问题,才能通过计算机求解。算法采用形式有多种,通常有自然语言、流程图、伪代码、N-S图、PAD图、程序语言等多种方法。能够用算法描述的问题称为可形式化的问题。由于算法仅仅是程序开发过程中的一个逻辑步骤,因而采用何种方法应考虑自己掌握的熟悉程度,但最终都要转换成计算机所能识别的程序。

正确的算法具备三个条件:

(1)每个逻辑步骤有可以实现的语句来完成;

(2)每个步骤间的关系是惟一的;

(3)算法要能终止(防止死循环)。

1. 算法的基本特征

算法是一个有穷规则的集合,这些规则确定了解决某类问题的一个运算序列。对于该

类问题的任何初始输入值，它都能机械地一步一步地执行计算，经过有限步骤后终止计算并产生输出结果。归纳起来，算法具有以下基本特征：

(1)输入：有零个或多个数据的输入。

(2)输出：有一个或多个数据的输出。

(3)有穷性：一个算法应包含有限的操作步骤而不能是无限的。

(4)确定性：算法中每一个步骤应当是确定的。无论算法在何种环境下实现，都应该得到相同的、确定的结果。

(5)有效性：算法中每一个步骤应当能有效地执行，并得到确定的结果。

2. 算法的表示

这里例举四种表示算法的方法。

(1)用自然语言表示算法

自然语言可以是中文、英文、数学表达式等。用自然语言表示通俗易懂，缺点是可能文字过长，不太严格，表达分支和循环的结构不很方便。

【例 1.5】 输入两个数，求其中的最大值。算法可表示如下：

① 设两个数为 x 和 y，最大数为 z；

② 输入两个数给 x 和 y；

③ 如果 x 大于或等于 y，则最大数 imax 的值等于 x 的值，否则为 y 值；

④ 输出最大值，结束算法。

【例 1.6】 求数列 1＋2＋…＋m 的值 N，当 N＞10000 时结束。算法可表示如下：

① n＝0；

② m＝0；

③ m 加 1；

④ N 加 m；

⑤ 判断 N 是否大于 10000，如果满足关系结束；不满足关系继续执行③。

(2)用流程图表示算法

流程图是用一些图框来表示各种操作。优点是直观形象，简单易于理解，便于修改和交流。ANSI C 规定了一些常用的符号，如图 1－7 所示。

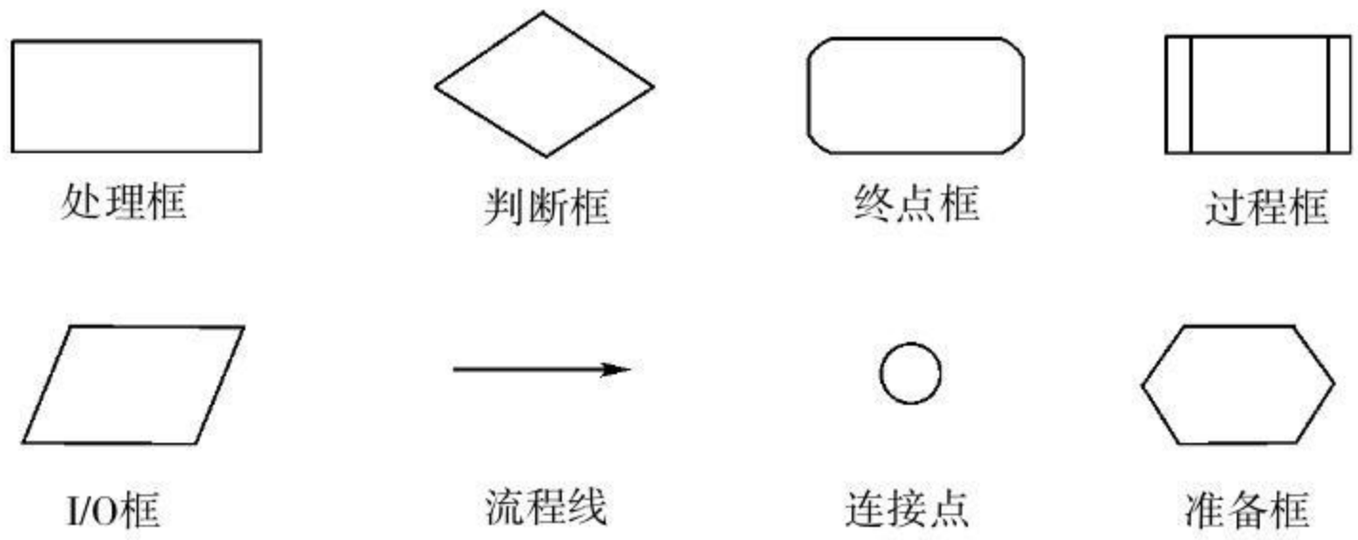


图 1－7　标准流程图符号

【例 1.7】 用框图描述如下函数的求解过程，如图 1－8 所示。

$$y=\begin{cases}5 & x\geqslant 0\\ -5 & x<0\end{cases}$$

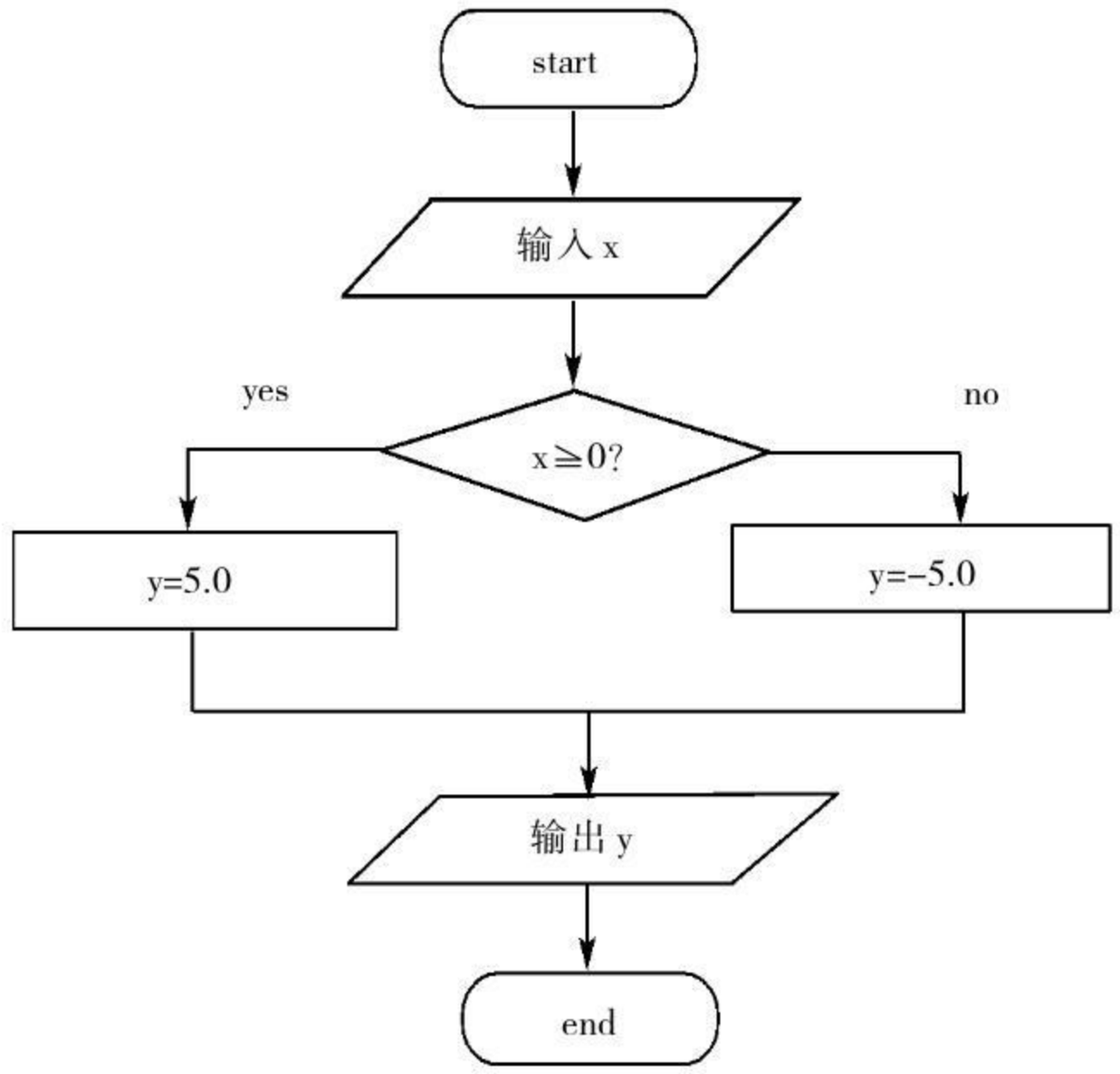


图1-8 例1.7流程图

(3)用伪代码表示算法

伪代码是介于自然语言和计算机语言之间的文字和符号来描述算法的,它不用图形符号。因此,书写方便,格式紧凑,也容易懂,同时也便于向计算机语言算法的转换。

【例1.8】 用伪代码表示求10!的算法。

```
begin(开始)
   置t的初值为1(1⇒t)
   置i的初值为2(2⇒i)
   当(while)i<=10,执行(do)
   begin
        使t=t×I
        使i=i+1
   end
        打印t的值(print t)
end(结束)
```


(4)用计算机语言表示算法

计算机是无法识别流程图和伪代码形式,只有用计算机语言编写的程序才能被计算机执行。因此在用流程图和伪代码形式描述出算法后,还要将它转换成计算机语言程序。

【例1.9】 用C语言程序来描述求10!。

```
#include <stdio.h>
void main(void)
{
   long  i,t;
   t=1L;i=2L;      /*给变量赋初值*/
```

```
    while(i<=10L)
       {
          t=t*i; i=i+1;
       }
       printf("%ld\n",t);
}
```

重点：算法具有输入、输出、有穷性、确定性和可行性 5 个基本特征。

1.4　C 程序运行环境与学习方法

1.4.1　C 程序的上机步骤

从编写好一个 C 程序到完成运行，一般需要经过以下四个步骤。

1. 编辑

所谓编辑，包括以下内容：

① 将源程序逐个字符输入到计算机内存；

② 修改源程序；

③ 将修改好的源程序保存在磁盘文件中，这些文件以“.c”为后缀名。编辑的对象是源程序，它是以 ASCII 代码的形式输入和存储的，不能被计算机执行。

任何一种编辑器都可以完成这项工作。例如：Windows 平台下的“记事本”；DOS 环境下的“EDIT”行编辑命令；TC 集成开发环境；VC 集成开发环境等。

2. 编译

编译就是将已编辑好的源程序（已存储在磁盘文件中）翻译成二进制的目标代码。在编译时，还要对源程序进行语法检查，如发现有错误，则在屏幕上显示出错信息。此时应重新进入编辑状态，对源程序进行修改后，再重新编译，直到通过编译为止。编译后生成后缀为“obj”的文件。

应当指出，经编译后得到的二进制代码还不能直接执行，因为每一个模块往往是单独编译的，必须把经过编译的各个模块的目标代码与系统提供的标准模块（如 C 语言中的标准函数库）连接后才能运行。

3. 连接

将各模块的二进制代码与系统标准模块经连接处理后，得到具有绝对地址的可执行文件，其后缀为“.exe”。

4. 运行

当编译生成“.exe”文件后，它是计算机能直接执行的文件，可以不在 TC 或 VC 集成环境下运行。

图 1-9 表示出一个 C 语言程序经过编辑、编译、连接、运行的全过程。

1.4.2　选择编程工具

C 语言的运行环境很多，目前常用的主要有两种：一种用 Turbo C 集成环境下运行；另

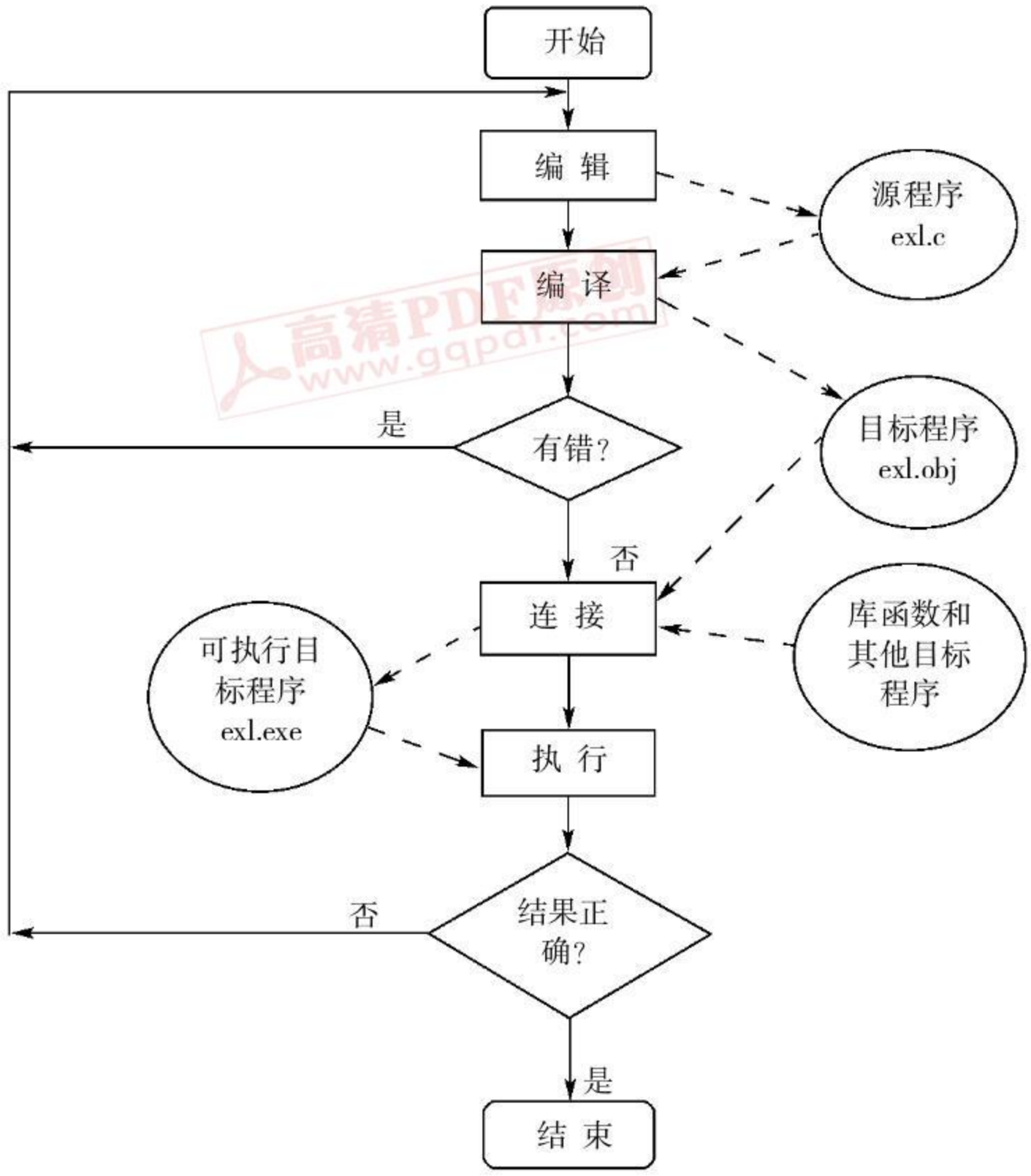


图 1-9 上机全过程

一种是在 VC++6.0 集成环境下运行。下面对这两个环境作些简单介绍。

1. Turbo C2.0 环境

Turbo C 是在微型机上广泛使用的编译程序。它具有方便、直观、易用的界面和丰富的库函数。它向用户提供一个集成环境,把程序的编辑、编译、连接和运行等操作全部集中在一个界面上进行,使用十分方便。

为了能使用 Turbo C,必须先将 Turbo C 编译程序装入磁盘某一目录下,例如放在 C 盘根目录的 TC 子目录下。

(1)启用 Turbo C 程序

通常 TC 执行的图标放在桌面上,直接点击即可。如果桌面上没有 TC 图标,可以打开 C 盘的 TC 目录,从中找到 TC.EXE 文件,双击即可。

如果打开正确,则出现如图 1-10 所示界面,这时按下 ESC 键,使光标进入编辑区,即可输入源程序了。

TC 环境界面的“主菜单”有 8 个菜单项,分别为:

File Edit Run Compile Project Options Debug Break/Watch

用户可以通过以上菜单项来使用 TC 环境提供的各项功能。上述 8 个菜单项分别代表:文件操作、编辑、运行、编译、项目文件、选项、调试、中断/观察等功能。

用键盘上的“←”和“→”键可以选择菜单条中所需要的菜单项,被选中的项以“反相”形式显示(例如主菜单中的各项原来以白底黑字显示,选中时改为以黑底白字显示)。此时若

图 1-10　Turbo C 集成环境

按回车键，就会出现一个下拉菜单。可以用“↓”键选择所需要的项。例如选择“File New”处，并按回车键，表示要建立一个新文件。

在这个环境下，一个程序从编辑、编译、连接直到运行出正确结果为止。

(2)常用的功能键

在调试程序中，通常可以用两种方法：一种用工具中提供的菜单项来完成(这里不再叙述)；还有一种就是用功能键。

在 Turbo C2.0 集成开发环境下常用到 F1～F10 功能热键，其功能见表 1-2 所列。

表 1-2　Turbo C2.0 集成开发环境下的功能热键

热键	功　　能
F1	激活帮助窗口，显示于当前光标所在位置有关的操作提示信息
F2	将当前文件以指定的文件名存盘
F3	装入指定文件
F4	将程序执行到光标所在的暂停
F5	缩放当前窗口
F6	取消亮条
F7	调试程序，执行单步操作，可进入被调用函数
F8	调试程序，执行单步操作，不进入被调用函数
F9	编译、连接源程序，生成可执行文件
F10	激活主菜单
Esc	返回上一级菜单

除上述功能键外，还用到几个组合键：

Ctrl＋F9：编译、连接和运行。

Alt＋F5：切换到屏幕看结果。

Alt＋菜单首字母：弹出菜单(例如 Alt＋F，弹出 File 菜单)。

2. VC＋＋6.0 环境

Visual C＋＋6.0(VC＋＋6.0)是 C/C＋＋的集成开发环境。C 程序的编写、运行过程可以分为 2 个阶段：

◀创建一个C源文件,输入源文件;

◀进行编译、连接、运行。

具体操作过程:

(1)选择“File/New”命令后,进入如图1-11所示界面;

(2)在图1-11界面中,选择“File/C++Source File”,在“File”文本框中输入程序名,选择源文件存放的位置,点击“ok”确定后,进入编辑区,即可输入源程序。输入源程序后的界面,如图1-12所示。

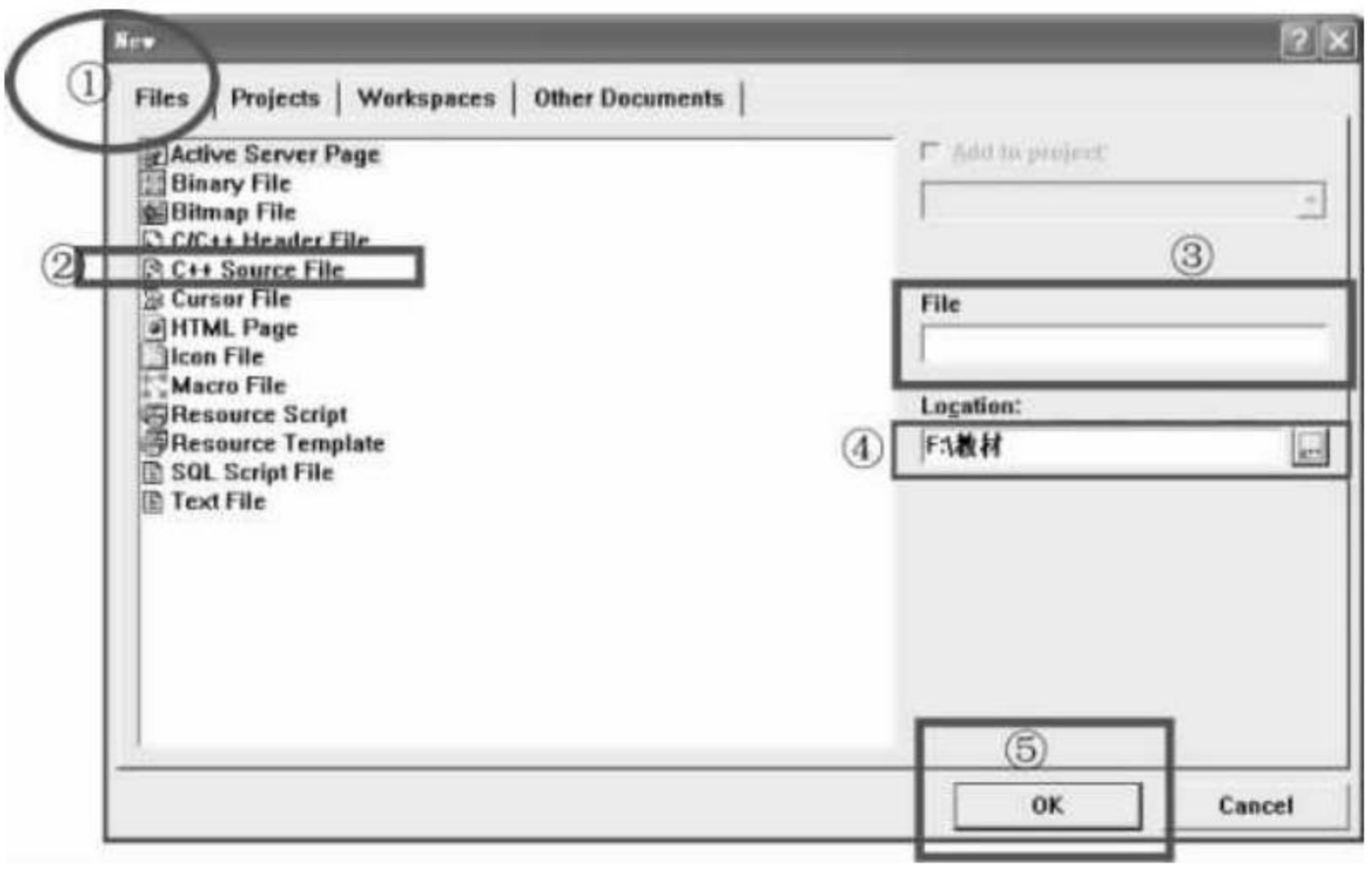


图1-11 建立C源程序对话框

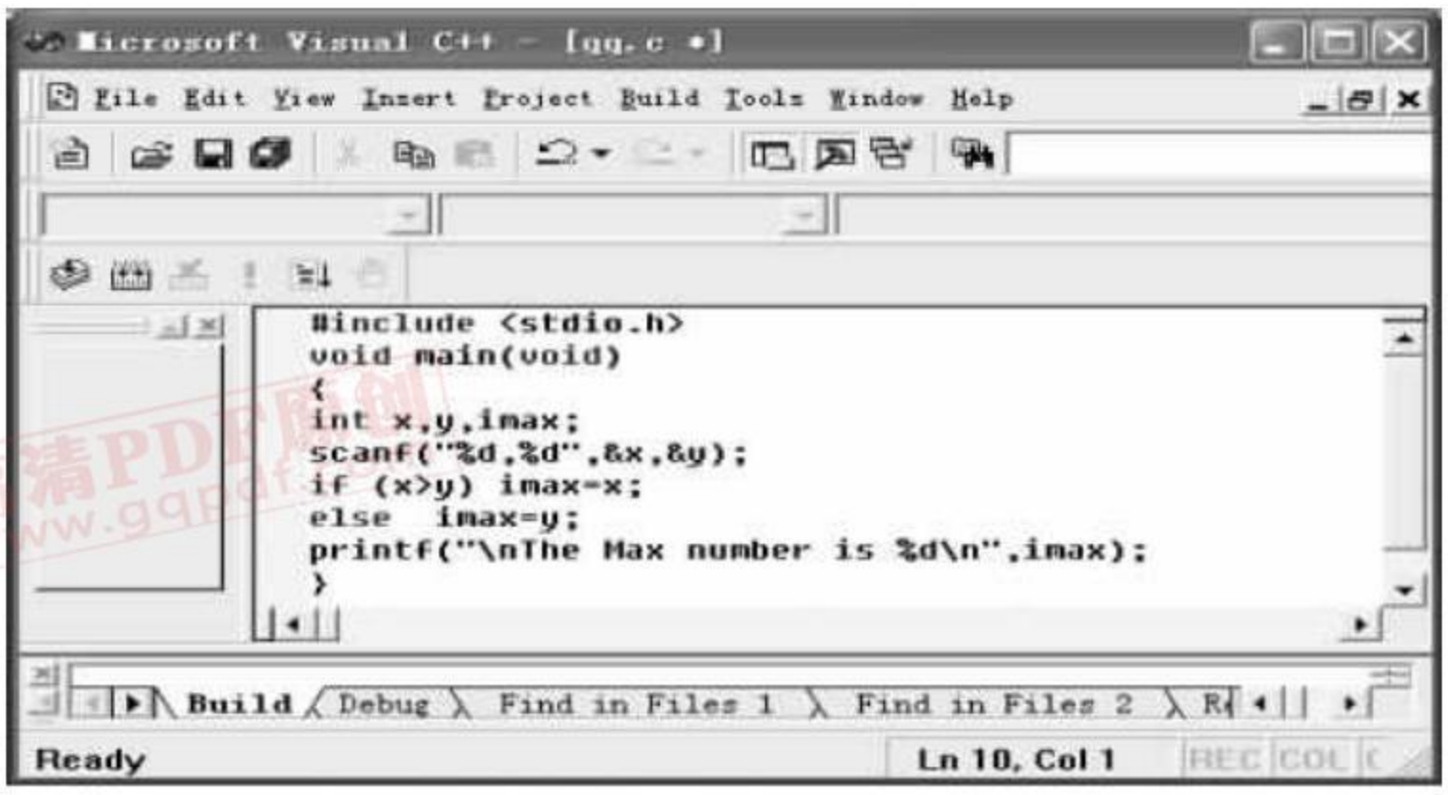


图1-12 编辑、调试C程序窗口

(3)编译、连接和运行

◀编译 选择“Build/compile qq.c”命令,编译结果显示在输出区中,如果没有错误,则生成qq.obj文件。

◀连接 选择“Build/ Build qq.exe”命令,连接结果显示在输出区中,如果没有错误,则生成qq.exe文件。

◀运行 选择“Build/ Execute qq.exe”命令,直到调试正确结果为止。

至此，一个简单的 C 程序的编写、调试过程结束。在编译、连接、运行时，可以通过菜单完成，也可以通过工具栏的按钮来实现。

重点：C 语言程序的上机环境常用的有 TC2.0 和 VC＋＋6.0；其编辑和运行步骤可以不在集成环境下进行。

1.4.3　学习 C 语言的方法

C 语言对于初学者来说，感觉难度比较大。要想学好 C 语言需要经历三个步骤：读程序、写程序和积累功能代码段。

1. 读程序

读程序是掌握程序设计思想、掌握编程方法的一个捷径。

要想读懂一个程序，必须具备一定的语言基础知识。在这个阶段有没有程序的设计思想并不重要，只要具备一定的语法基础即可。

学习一门语言并不需要刻意去记条条框框的语法，看程序代码时，遇到不明白的地方再去查相关的资料，补充基础知识再配合源程序的思路，这时的理解才能最深刻。

2. 写程序

刚开始写程序，不要奢望一下子就能写出很出色的程序来，“万丈高楼平地起”，编程贵在动手，只要动手去写就可以了。在编程的过程中要循序渐进，开始时写一些功能简单、代码短小的程序，然后在些基础上进行扩充，直到掌握编程的思想和方法。

3. 积累功能代码段

积累也是一个很重要的环节，平时把自己编好的程序或自己读懂的程序分类保存起来，建一个属于自己的代码库，需要相关功能时，就到自己的代码库中查找，这样既提高了编码的效率，又提高了正确性。随着时间和知识的积累，你已掌握了 C 语言的编程方法和技巧。

提示：学好 C 语言的密诀是多读程序，勤写程序，学会整理和规纳。

1.5　例题精解

1. 在 C 语言程序中，main()函数的位置_____。

A)必须作为第一个函数　　B)必须作为最后一个函数

C)可以放在任意位置　　D)必须放在它所调用的函数之后

【解析】C 语言规定，一个完整的 C 程序必须有且仅有一个 main()函数。按照 C 程序的书写风格，main()函数可以放在任意位置上。但是程序执行时总是从 main()函数开始，直到 main()函数结束。其他函数只有在被 main()函数调用或嵌套调用时才能被执行。

【答案】C

2. 结构化程序设计所规定的 3 种基本控制结构是_____。

A)顺序结构、选择结构、循环结构　　B)输入、输出、处理

C)主函数、子函数、函数　　D)单入口、单出口、无死语句

【解析】结构化程序设计是由 3 种基本结构组成，分别是顺序结构、选择结构、循环结构。

【答案】A

3. C语言源程序文件的扩展名是_____。

A). cpp　　B). c　　C). obj　　D). exe

【解析】C语言源程序的扩展名为.c,而.cpp是c++程序的扩展名。.obj是目标文件扩展名;.exe是可执行文件的扩展名。

【答案】B

4. C程序开发过程,一般要经过编辑、_____、连接运行4个步骤。

【解析】C语言是以编译方式实现的高级语言。C程序的实现必须要使用某种C的的编译器进行编译,编译是C开发过程中不可缺少的环节。

1.6　本章小结

本章简要介绍了C语言的发展史,C语言的功能和特点;并从程序设计的角度,介绍了程序、程序设计的概念,程序设计的方法和算法。

本章通过几个简单案例,分析了C程序的基本构成和书写风格,总结了程序的组成、函数的组成、语句的组成以及C程序的基本结构特点。

C语言的注释符是以"/*"开头并以"*/"结尾。在"/*"和"*/"之间的即为注释内容。程序编译时,不对注释作任何处理,注释可出现在程序中的任何位置。

C语言的上机环境现在常用的主要有两种:Turbo C和Visual C++6.0。以前主要用TC环境,而现在由于全国计算机等级考试将运行环境由TC改为VC++6.0,现在很多学校也随之做了更改,为了便于学生今后参加全国计算机等级考试,本教材也将程序的运行环境由TC改为VC++6.0。为了过渡,本章将TC和VC++6.0运行环境都做了介绍。

学习C语言一般来说有相应的难度,但功夫不负有心人,只要下功夫,什么样的堡垒都可以攻破。本章从三个方面规纳了学习C语言的方法,供学习者参考。

习　题

一、单项选择题

1. C程序的基本单位是_____。

A)函数　　B)标识符

C)表达式　　D)语句

2. 算法具有五个特性,以下选项中不属于算法特性的是_____。

A)有穷性　　B)简洁性

C)可行性　　D)确定性

3. C程序是由_____构成的。

A)主函数与子函数

B)一个主函数与一个其他函数

C)主函数与若干子函数

D)主函数与子程序

4. 在 C 语言程序中，main()函数的位置_____。

A)必须作为第一个函数

B)必须作为最后一个函数

C)可以放在任意位置

D)必须放在它所调用的函数之后

5. 下面叙述中不正确的是_____。

A)注释说明只能位于一条语句的后面

B)注释说明被计算机编译系统忽略

C)注释说明必须括在“/ * … * /”之间

D)注释符“/”和“ * ”之间不能有空格

6. 下面说法中，正确的是_____。

A)主函数名是由程序设计人员按照“标识符”的规则选取的

B)分号和回车符号都可以作为一个语句的结束符号

C)在程序的任何地方都可以插入一个或多个空格符号

D)程序的执行总是从源程序清单的第 1 行开始执行

7. 以下说法中正确的是_____。

A)C 语言程序总是从第一个定义的函数开始执行

B)C 语言程序总是从 main()函数开始执行

C)C 语言程序中的 main()函数必须放在程序的开始部分

D)一个 C 函数中只允许一对花括号

8. C 程序的上机步骤有编辑、编译、连接和运行，下面可以不在运行环境下(TC 或 VC)进行的是_____。

A)编辑、编译　　B)编译、连接

C)编辑、运行　　D)连接、运行

二、填空题

1. 用高级语言编写的源程序必须通过_____程序翻译成二进制程序才能执行，这个二进制程序称为_____程序。

2. C 程序的执行总是由_____函数开始，并且在_____函数中结束。

3. C 程序的语句结束符是_____。

4. 一个完整的 C 程序至少有一个_____函数。

5. C 程序执行文件的扩展名是_____。

6. 结构化程序设计由三种控制结构组成，即_____、_____、_____。

7. 输入/输出函数存放在_____文件中。

第2章 数据类型、运算符与表达式

【教学提示】

本章主要介绍C语言的基本数据类型、常量的书写方法和变量的定义、赋值、初始化和使用方法，基本运算符的运算规则和表达式的构成方法，以及不同类型数据运算的类型转换规则，为后续章节的学习奠定基础。

【核心概念】

关键字 标识符 数据类型 常量 变量 运算符 表达式 优先级

2.1 C语言基础

C语言是一种结构程序设计语言，由C语言编写的程序是由各种不同的词法记号构成的。所谓词法记号，是指程序中具有独立含义的不可进一步分割的单位。C语言的词法记号可分成5类：

(1)关键字

(2)标识符

(3)常量

(4)运算符

(5)分隔符

2.1.1 关键字

关键字，也称为保留字，是C语言中预定义的符号，它们有固定的含义，用户定义的标识符不允许与它们冲突。C语言中的关键字是由小写字母构成的字符序列，见表2-1所列。

表2-1 C语言中的关键字

关键字	语义	关键字	语义	关键字	语义	关键字	语义
auto	自助	break	中断	case	情况	char	字符
const	常量	continue	继续	default	缺省	do	做
double	双精度	else	否则	enum	枚举	extern	外部
float	浮点	for	对于	goto	转向	if	如果
int	整数	long	长整型	register	寄存器	return	返回
short	短	signed	有符号	sizeof	字节数	static	静态
struct	结构	switch	开关	typedef	类型定义	union	共用
unsigned	无符号	void	空	volatile	可交的	while	当

C 语言 32 个关键字，根据其作用，可将其分为数据类型关键字、控制语句关键字、存储类型关键字和其他关键字四类：

1. 数据类型关键字有（12 个）：int、char、float、double、long、short、void 、signed、unsigned、enum 、struct、union。

2. 控制语句关键字（12 个）：

（1）循环语句：for、do、while、break、continue；

（2）条件语句：if、else、goto；

（3）开关语句：switch、case、default；

（4）返回语句：return。

3. 存储类型关键字（4 个）：auto、extern、register、static。

4. 其他关键字（4 个）：const、sizeof、typedef、volatile。

注意：关键字一律用小写字母。

2.1.2　标识符

标识符是指用来表示变量名、函数名、数组名、类型名、文件名等有效字符序列。

在 C 语言中各种名称都是由标识符来表示的，C 语言中没有标准标识符的概念，main 可以看成为唯一的标准标识符，它被编译程序预定义为主函数的名字。通俗地说，标识符就相当于一个人的名字。作为 C 语言的标识符，ANSI C 规定必须满足以下规则：

（1）所有标识符的第一个字符必须是以字母（a～z，A～Z）或下划线（_）打头；

（2）标识符的其他部分必须由字母、下划线或数字（0～9）组成；

（3）标识符的有效长度，一般不超过 32 个字符。在实际应用中，标识符的选择不宜太长，一般按照“见名知义”、“常用取简”的原则，使之一目了然，以增加程序的可读性。

（4）标识符不能与 C 中的关键字同名。

以下是合法的标识符：

a　sum　x2　a_bc　_num

而下列标识符是不合法的：

*day　3xy　－sting1

C 语言还规定，标识符中同一个字母的大写与小写被看作是不同的字符（即标识符区分大小写）。这样，a 和 A，ABc 和 abc 是互不相同的标识符。

2.1.3　C 语言的数据类型

C 语言中，为解决具体问题，要采用各种类型的数据，数据的类型不同，它所表达的范围、精度和所占据的存储空间均不相同，对不同类型规定了不同运算。

C 语言提供了三大数据类型，即基本类型、复合类型和地址类型。

（1）基本类型只代表单个数据。如：char、int、float、double、void。

（2）复合类型由基本类型组合而成，可代表一批数据。如：struct、union、enum。

（3）地址类型可直接表示内存中的地址。如：指针类型。本节首先讨论基本数据类型。

1. 整型(int)

C语言中,整型数据根据其长度的不同,分为基本整型、短整型、长整型和无符号型。

在不同的C编译环境中整型数据所占据的内存空间的长度(即字节数)有所不同,它们所占位数和数的范围,见表2-2所列。

表2-2 各种类型数据所占位数及数的范围

数据类型	类型说明符	Turbo C2.0		Visual C++6.0	
		字节数	取值范围	字节数	取值范围
基本整型	int	2	−32768～32767	4	−2147483648～2147483647
短整型	short[int]	2	−32768～32767	2	−32768～32767
长整型	long[int]	4	−2147483648～2147483647	4	−2147483648～2147483647
无符号整型	unsigned[int]	2	0～65535	4	0～4294967295
无符号短整型	unsigned short[int]	2	0～65535	2	0～65535
无符号长整型	unsigned long[int]	4	0～4294967295	4	0～4294967295

整数在计算机内容按该数的二进制补码形式存放。其中首位是符号位,符号位用“0”表示正数,用“1”表示负数。对于正整数,其原码、反码、补码相同;负整数的补码为数的原码除符号位,其余位按位取反(原0改为1,原1改为0)后再加1。

无符号数在内存中存放时二进制最高位不是符号位,而有符号整数在内存中以补码表示,其最高位是1时表示负数,最高位是0时表示非负数。

2. 浮点型(float和double)

浮点型用来表示实型数据,可分为单精度浮点型和双精度浮点型,分别用float和double表示。C语言中的实型数,在机器内部是以浮点数形式存储的,数值都是近似的,引入double型后,可以改善计算的精度。一般编译系统为浮点型数的分配参数,见表2-3所列。

表2-3 实型数据所占位数及数的范围

数据类型	类型说明符	占用字节数	数值范围
单精度浮点型	float	4	$-3.4e^{38}\sim3.4e^{-37}/3.4e^{-37}\sim3.4e^{38}$
双精度浮点型	double	8	$-1.7e^{308}\sim1.7e^{-307}/1.7e^{-307}-1.7e^{308}$

不同的编译系统,有效数字的位数是有区别的。

3. 字符型(char)

字符型数据代表ASCII码字符集中的一个字符,在程序中用单引号括起来。一个字符型数据占用一个字节的长度,它实质上是个8位的整型量,取值范围−128～127。因此,字符型和整型关系非常密切,可把字符型看做一种特殊的整型,事实上,字符型数据和整型数据之间经常混合使用。

4. 空类型(void)

空类型表示无返回值的函数或无定向指针。

注意:ANSI C 标准没有具体规定各类数据在内存中所占的字节数,由各 C 编译系统自行决定。

2.2　常　量

常量是指程序在运行时其值不能改变的量。常量包括整型常量、浮点型常量、字符型常量、字符串常量和符号常量。

1. 整型常量

整型常量即整常数,由一个或多个数字组成,可以带正负号。整型常量根据表示形式可以分为十进制、八进制、十六进制。

(1)十进制整型常量包括:0～9 数字、±号的数字串。如:123,0,－90。

(2)八进制整型常量是以数字 0 开头,包括 0～7 数字串。如:024 表示八进制数 24,即 $(24)_8$,其值为:$2\times8^1+4\times8^0$,等于十进制数 20。

(3)十六进制整型常量是以 0x 或 0X 开头,包括 0～9 数字、a～f 的数字和字母串。如:0x13a,代表十六进制数 13a,即 $(13a)_{16}=1\times16^2+3\times16^1+10\times16^0=256+48+10=314$。

下列是合法的整常量:

12345　－523　0　027　0x29　0X3d7

在整型常量后跟有字母 l 或 L 时,表示该整型常量是长整型常量。如:

53458L　　0X39F6DL

2. 实型常量

实型常量即浮点型常量。实型常量只能以十进制形式表示。共有两种表示形式:小数表示法和指数表示法。

(1)小数表示法:由数字和小数点组成,如 0.256、3.14 等。

(2)指数表示法:由数字、小数点和指数部分组成。如用 3.14e＋5 表示 3.14×10^5,1.23E－4 表示 0.000123 等。

合法的指数形式应做到字母 e(或 E)之前必须有数字,后面指数必须为整数。因此,E7、1.3e1.5 都不是合法的实型常量。

3. 字符常量

字符型常量是用单引号括起来的一个字符。如'a'、'A'、'2'、'*'、'?'、'␣'(表示空格字符)等,这里'a'和'A'是不同的字符型常量,在 C 中每个字符常量占 1 个字节,并有确定的值,即该字符的 ASCII 码值。如'a'的 ASCII 码值为 97,而'A'的 ASCII 码值为 65,'2'的 ASCII 码值为 50 等。

除了以上形式的字符常量外,对于常用的但却难以用一般形式表示的不可显示字符,C 语言提供了一种特殊形式的字符常量,就是以反斜杠“\”引导的字符序列,这种表示方式的字符称为转义字符,如'\n'表示一个换行符。转义字符常用来表示控制字符和系统占用字符,见表 2－4 所列。

说明：

① 转义字符常量，如'\n'、'\101'、'\141'只代表一个字符。

② 对于整型常量可以用八进制形式表示，此时必须以数字0开头，如058就表示八进制数58。而在转义字符中，反斜线后的八进制数可以不用0开头。如：'\101'中的101就是八进制数，该转义字符代表的就是字符常量'A'。

③ 十六进制的转义字符，反斜线后必须要有'x'标记，如：'\x41'代表字符常量'A'。

表2-4 转义字符表

字符形式	含义	字符形式	含义
\n	换行	\f	换行但不回车
\t	横向跳格(Tab)	\\	反斜杠\
\v	竖向跳格	\’	单引号
\b	退格	\ddd	1到3位8进制数所代表的字符
\r	回车	\xhh	1到2位16进制数所代表的字符

注意：字符常量只能包括一个字符，如'AB'是不合法的；字符常量区分大小写字母，如'A'和'a'是两个不同的字符常量。

4. 字符串常量

字符串常量是指用双引号括起来的一串字符，如"a"、"I am a student"、"345"等。请注意'a'和"a"的区别，它们在计算机内的存储方式是不同的。C规定：字符串常量在内存中存储时，系统自动在其尾部追加一个字符串结束标志(即字符'\0')。'\0'表示ASCII码值为0的字符，它既不显示也不会引起任何控制动作。

所以，字符串常量"a"在内存中占用两个字节，而字符常量'a'在内存中占一个字节。字符串"China"的存储方式为：

C	h	i	n	a	\0

在C中可以用字符变量来存储一个字符常量，但是没有专门的字符串变量存储字符串，字符串的存储需要使用字符数组，这将在以后的章节中介绍。

字符常量与字符串常量的区别：

① 定界符不同：字符常量使用单引号，而字符串常量使用双引号；

② 长度不同：字符常量的长度固定为1，而字符串常量的长度可以是0，也可以是某个整数；

③ 存储要求不同：字符常量存储的是字符的ASCII码值，而字符串常量，除了要存储有效的字符外，还要存储一个结束标志'\0'。

5. 符号常量

C语言中用一个标识符(符号)代表一个常量，称为符号常量。符号常量名用大写字母书写。符号常量的定义形式为：

#define 符号常量名 常量

其中＃define 也是一条预处理命令(预处理命令以"＃"开头),称为宏定义命令,其功能是把该符号常量名定义为其后的常量值。一经定义,以后在程序中所有出现该符号常量名的地方均代之以该常量值。例如:

```
＃include＜stdio. h＞
＃define PI 3. 14159
void main(void)
{
    float s,r;
    r＝5;
    s＝PI＊r＊r;
    printf("%f\n",s);
}
```

定义 PI 为符号常量名,在程序中可用符号常量名 PI 代表数值 3. 14159。

说明:

① 符号常量在使用之前必须先定义,符号被定义后,在程序运行过程中,其值不可改变,在程序中符号常量名就代表定义时给出的数值。

② 习惯上,符号常量的标识符用大写字母表示。

2.3　变　量

在 C 语言中,通常用变量来保存程序运行过程中输入的数据、中间运算结果、最终运算结果等。一个变量由一个名字来标识,即变量名,C 语言中的变量名必须符合标识符的命名规则。

每个变量的使用时提供两个信息:

① 变量的地址。即操作系统为变量在内存中分配的若干内存的首地址;

② 变量的值。即变量在内存中所分配的那些内存单元中所存放的数据。

C 语言中,变量在内存中开辟地的大小由数据类型来决定,由于 PC 机中规定一个地址单元存放一个字节,不同的数据类型的变量,为其分配的地址单元数是不一样的,因而变量在使用之前必须加以说明其类型,然后才能使用。

变量类型说明的一般形式为:

＜类型标识符＞　变量名表;.

其中:

① 类型标识符

分:基本类型标识符:int、char、float、double

　　修饰类型标识符:unsigned/signed、short/long

② 变量名表:是用逗号分隔的标识符。

例如:

```
int i;                  /＊i 为整型变量,在内存 TC 分配 2 个字节,VC 分配 4 个字节 ＊/
short a,b,c;            /＊ a,b,c 为短整型变量,每个变量在内存分配 2 个字节 ＊/
```

```
float x,y;                  /* x,y为单精度型变量,每个变量在内存分配4个字节 */
double ave,sum;             /* ave,sum为双精度型变量,每个变量在内存分配8个字节 */
char ch1,ch2;               /* ch1,ch2为字符型变量,每个变量在内存分配1个字节 */
unsigned long f1,f2;        /* f1,f2为无符号长整型变量,每个变量在内存分配4个字节 */
```

上述变量被说明后,根据其类型的不同,拥有不同大小的存储单元。

在书写变量定义时,应注意以下几点:

(1)允许在一个类型说明符后,定义多个相同类型的变量。各变量名之间用逗号间隔。

(2)最后一个变量名之后必须以“;”号结尾。

(3)变量定义必须放在变量使用之前,一般放在函数体的开头处。

注意:变量名是标识符,因此其大小写是有区别的。Abc 和 aBc 是两个不同的变量名。习惯上,变量名用小写字母表示,以增加可读性。

2.4 基本运算符与表达式

2.4.1 C运算符概述

运算是对数据的加工,最基本的运算形式常常可以用一些简洁的符号来记述。这种在描述数据加工时,用来表示各种运算的符号称为运算符。用运算符将运算对象连接形成的运算式就是运算表达式,简称表达式。表达式本身就是描述数据加工的一种方法,只不过它所描述的是比较简单的数据加工过程。

C语言的运算符是一种向编译程序说明一个特定的数学或逻辑运算的符号,在应用于表达式中的变量或其他变量时,将会触发某一运算。运算符必须有运算对象。C语言中的运算符的运算对象可以是一个,称单目运算符;运算对象也可以是两个,称双目运算符;运算对象还可以是三个,称三目运算符。单目运算符若放在运算对象的前面称为前缀单目运算符;若放在运算对象的后面则称为后缀单目运算符。双目运算符都是放在两个运算对象的中间。三目运算符是夹在三个运算对象之间的。

在表达式中若出现多个运算符,计算表达式值时,就会碰到哪个先算,哪个后算的问题,我们把这个问题称为运算符的优先级。计算表达式值时,必须考虑运算符的优先级别,优先级高的要先算。

注意:在复杂的表达式中,用圆括号括住的部分要先算,其优先级别高于任何运算符。若在圆括号中又有圆括号,则内层圆括号优先于外层圆括号。

C语言中,同级别的运算符还规定了结合性,即运算时是自左向右进行的,称为左结合;运算时是自右向左进行的,称为右结合。

注意:C语言中的运算符都是键盘上的符号或若干符号的组合,书写或输入时不要出错。

2.4.2　算术运算符和算术表达式

1. 算术运算符

算术运算符是对数据进行简单的算术运算。要注意字符型数据也可看成整数，参加算术运算。

基本的算术运算符有：

＋　正值运算符，或加法运算符。如＋4，10＋7。

－　负值运算符，或减法运算符。如－4，10－7。

*　乘法运算符。如 10 * 3.14。

/　除法运算符。如 10/7，两个整型量相除，自动向下取整。

%　模运算符，或称求余运算符。如－10%7，余数的符号同被除数。

有关算术运算符的运算对象、运算规则与结果、结合性见表 2-5 所列。

表 2-5　算术运算符

对象数	名称	运算符	运算规则	运算对象	运算结果	结合性
单目	正 负	＋ －	取原值 取负值	整型或实型	整型或实型	自右向左
双目	加 减 乘 除	＋ － * /	加法 减法 乘法 除法	整型或实型	整型或实型	自左向右
	模	%	整除取余	整型、字符型	整型	

单目正(＋)运算并不改变运算对象的值，所以很少使用。单目负(－)运算是取运算对象的负值。例如，＋7 的结果是正整数 7，－7 的结果是负整数 7。

双目加(＋)、双目减(－)和乘法(*)运算同普通算术运算中的加法、减法和乘法相同。例如，3.2＋4.3 的结果是 7.5；3.2－4.3 的结果是－1.1；3.2 * 2 的结果是 6.4。

说明：

(1)除求余运算符外，其余运算符可以用于整型数和实型数运符。加、减法运算符还可用于字符运算。

(2)表达式中的“*”不能省略。

(3)“/”运算符可以实现整型数和实型数的除法运算。当其左右两边的操作数都是整型数时，则运算结果自动向下取整。如：1/2 结果为 0，9/4 结果为 2；而 1.0/2、1/2.0、1.0/2.0 的结果均为 0.5。

(4)“%”运算符要求两个运算对象均为整型或字符型数据，运算结果是整除后的余数，余数的符号取决于被除数的符号。

(5)双目运算符两侧操作数的类型要相同。如果一个运算符的两侧的数据类型不同，则先自动进行类型转换，使二者具有同一种类型，然后再进行运算。例如表达式 1/2.0，在进行除法运算之前，系统首先将操作数 1 由 int 类型转换成 double 类型，然后再进行除法运算。

2.算术运算符的优先次序

算术运算符的优先级：

(1)单目算术运算符优于双目算术运算符；

(2)“*，/，%”优于“+，-”；

(3)同级单目算术运算符的结合性满足右结合，同级双目算术运算符的结合性满足左结合。

3.算术表达式

由算术运算符和括号将运算对象(也称操作数)连接起来的、符合C语法规则的式子称为C算术表达式。运算对象包括常量、变量和函数等。例如，下面是一个合法的C算术表达式：

a*b/c-3.0+'a'

C语言规定了运算符的优先级和结合性，在表达式求值时，先按运算符的优先级别高低顺序执行，例如a+b*c，计算时先算出b*c的值，然后再进行a与b*c的值相加运算。如果在一个运算对象两侧的运算符的优先级别相同，例如a+b-c，则按规定的“结合方向”处理。由于同级双目算术运算符满足左结合性，即先左后右，因此先执行a+b的运算，再执行减c的运算。

算术表达式的类型可能是整型、单精度实型或双精度实型。

使用和理解算术表达式时，要注意下列几点：

(1)字符型常量、字符型变量可以在算术表达式中使用。

(2)C语言允许各类数值型数据之间进行混合运算，在进行混合运算时，不同类型的数据要先转换成同一类型，然后进行运算(转换的规则见2.5节)。

例如：设a=2，c='a'，f=3.0

```
a+2-a*6/f+c%a        /*此式为算术表达式，由常量和变量构成，其值为1.0*/
```

2.4.3 自增自减运算符

自增(++)、自减(--)运算符属单目运算符，常用于整型、字符型、指针型以及数组的下标等数据进行算术运算，运算的结果仍是原类型。由于运算结果将存回运算对象，所以这两个运算符的运算对象只能是变量。用自增、自减运算符和运算对象组成表达式时，运算符可以放在运算对象的前面(前缀)，也可以放在运算对象的后面(后缀)，前缀和后缀出现在不同位置，其功能略有不同。

自增、自减运算符的运算对象、运算规则与结果、结合性见表2-6所列。

表2-6 自增自减运算符

对象数	运算符	前置	后置	运算关系	运算对象	运算结果	结合性
单目	++ --	++a --a	a++ a--	a=a+1 a=a-1	整型或字符型或指针型或下标等	同运算对象的类型	自右向左

自增、自减运算符在使用中会经常出现，但要注意以下几点要求，以免产生意想不到的副作用。

(1)自增、自减运算符只能用于变量，而不能用于常量或表达式。如 2＋＋或－－(x＋y)都是不合法的。

(2)自增、自减运算符的结合方向是右结合，即“自右至左”。例如，－i－－，左边是单目减，即 i 的负值数，右边是自减运算符，若按左结合性，相当于(－i)－－，而－i 是一表达式，因而是不合法的。由于单目减与“－－”运算符优先级相同，而结合性为右结合，因此－i－－实际上等价于－(i－－)，所以－i－－是合法的。

自增、自减运算符常用于循环语句中，使循环变量自动加 1 或减 1；也可用于指针变量，使指针指向下一个地址(具体使用方法将在以后章节作详细介绍)。

(3)在表达式中包含自增或自减运算时，要遵循后缀运算符，先取值后自增，对前缀运算符，先自增后取值的原则。设 i 的值等于 3，如果有以下赋值语句：

```
j=i＋＋;
```

那么，显然先将 i 的原值取出赋给 j(j 的值等于 3)，然后 i 进行自加，执行完该语句后，i 的值等于 4。但是如果有以下赋值语句：

```
j=(i＋＋)＋(i＋＋)＋(i＋＋);
```

求值时先把 i 的原值 3 取出来，作为表达式中 i 的值，因此先进行 3 个 i 相加，j 得 9，然后再实现自加，i 的值变为 6。而对于以下赋值语句：

```
j=(＋＋i)＋(＋＋i)＋(＋＋i);
```

求值时先对整个表达式扫描，对 i 进行 3 次自加，i 得 6，然后进行 6＋6＋6 的运算，故 j 值是 18。

(4)当出现难以区分的若干个＋或－组成的运算符串时，C 语言规定，自左至右取尽可能多的符号组成运算符。

例如，设 i，j 均为 2，则：

i＋＋＋j　应理解成(i＋＋)＋j，不能理解成 i＋(＋＋j)；

i－－－j　应理解成(i－－)－j，不能理解成 i－(－－j)。

> 注意：＋＋(自加)和－－(自减)是 C 的一个特色，可以使程序清晰、简练，但用得不适当，也会产生副作用。因而在使用中，为了便于理解和减少出错，需要时可以加括号。

2.4.4　赋值运算符和赋值表达式

1. 赋值运算符

赋值运算符为双目运算符，赋值号左边必须是变量，赋值号右边是表达式。C 语言中的赋值运算符分为基本赋值运算符和复合赋值运算符。

(1)基本赋值运算符

基本赋值运算符记为“＝”，由“＝”连接的式子称为简单赋值表达式，其一般形式为：

变量＝表达式；

功能：将赋值运算符右侧表达式的值赋给左侧的变量。

例如：

```
int a;
a=2+3;      /* 先说明,执行时赋值 */
int a=2;    /* 说明变量同时赋值 */
```


在使用赋值运算符时应注意以下几个问题：

① C语言中的赋值运算符的作用是将一个数据赋给一个变量。如“a=100”的作用是执行一次赋值操作(或称赋值运算),把常量100赋给变量a。赋值运算符不是数学中的等号,不是进行两个数据的比较。在C语言中相等的比较是用“==”。

② 如果赋值运算符两边的数据类型不一致,要以左边变量的类型为准进行类型转换。

♦ 将实型数据(包括float、double)赋给整型变量时,舍弃实数的小数部分,不做四舍五入的操作。如a为整型变量,执行“a=12.75”的结果是使a的值为12。

♦ 将整型数据赋给单、双精度变量时,数值不变,但以浮点数形式存储到变量中,然后以实型数据的精度要求以0补足有效数字。例如,将12赋给float变量f,即f=12,先将12转换成12.000000,再存储到f中。

(2)复合赋值运算符

在赋值号前面加上算术运算符,可以构成复合赋值运算符。它专用于将某个变量和运算式进行指定算术运算后的结果赋予该变量。实际上,它是一般赋值表达式的简写。

有关复合赋值运算符的运算对象、运算规则与结果、结合性见表2-7所列。

表2-7 复合赋值运算符

对象数	名称	运算符	运算规则	运算对象	运算结果	结合性
双目	加赋值	+=	a+=b 相当 a=a+b	数值型	数值型	自右向左
	减赋值	-=	a-=b 相当 a=a-b			
	乘赋值	*=	a*=b 相当 a=a*b			
	除赋值	/=	a/=b 相当 a=a/b	整型	整型	
	模赋值	%=	a%=b 相当 a=a%b			

使用复合赋值运算符便于简化赋值语句,使程序精炼,也便于提高编译效率,并有利于编译程序,以便能产生质量更高的目标代码。

在使用中应注意:对于a*=b+c等价于a=a*(b+c),而不是等价a=a*b+c。

实际上,凡是双目运算符,都可以与赋值运算符一起组合成复合赋值运算符。C语言除了规定上述五种复合赋值运算符,还有以下五种关于位运算的复合赋值运算符：

<<=, >>=, &=, ^=, |=

2.赋值运算符的优先次序

赋值运算符的优先次序如下：

(1)赋值运算符的优先级别较低,它低于双目逻辑运算符但高于逗号运算符。

(2)基本赋值运算符和复合赋值运算符是同级别的,结合性满足右结合。

3.赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式,它的一般形

式为：

＜变量＞＜赋值运算符＞＜表达式＞

其求解过程为：将赋值运算符右边的“表达式”的值赋给左边的变量。赋值表达式的值就是被赋值的变量的值。例如，“a＝12”这个赋值表达式的值为 12（变量 a 的值也是 12）。赋值运算符右边的“表达式”也可以是一个赋值表达式。如：

a＝(b＝(c＝2))

由于赋值运算符是按照右结合性结合的，所以可以简写为 a＝b＝c＝2，它表示将赋值表达式的值 2 赋给变量 a，b，c。

赋值表达式也可以包含复合的赋值运算符，在使用时一定要注意其结合性原则，例如：

a－＝a＋＝a＊a－3

假如 a 的值为 2，则此赋值表达式的求解过程为：先计算 a＋＝a＊a－3，此时 a 的值为 3，后计算 a－＝3，此时 a 的值为 0。

将赋值表达式作为表达式的一种，使赋值操作不仅可以出现在赋值语句中，而且可以以表达式形式出现在其他语句（如循环语句）中，这是 C 语言灵活性的一种表现，在以后的学习中要加以体会。

例如，设 a＝2，c＝'a'，f＝3.0，则下列表达式都是赋值表达式：

f＊＝a＋3	算术表达式构成，赋值表达式的值为 15.0
d＝a＜＝c	关系表达式构成，赋值表达式的值为 1
d＝！(a＝＝0)	逻辑表达式构成，赋值表达式的值为 1
d＝e＝f＋1.5	赋值表达式构成，赋值表达式的值为 4.5

2.4.5　其他运算符及表达式

1. 逗号运算符和逗号表达式

逗号运算符(,)是 C 语言中一个比较特殊的运算符，它的作用是将若干个表达式连接起来，其一般形式为：

表达式 1，表达式 2，表达式 3，… 表达式 n

例如：3＋5，4＊3；

这样用逗号把若干独立的运算表达式结合成为一个表达式称为逗号表达式。

又如：a＝3，b＝5，c＝a＊b；也是一个逗号表达式，它是由三个独立的表达式结合而成的。

说明：

(1)逗号表达式的求解过程是：先计算表达式 1 的值，再计算表达式 2 的值，…，一直计算到表达式 n 的值。整个逗号表达式的值是最后一个表达式 n 的值。

例如：

```
i＝5,j＝6,k＝7;          /＊ 整个逗号表达式的值是 7 ＊/
x＝5＊2, x＊4;           /＊ 整个逗号表达式的值是 40，x 的值是 10 ＊/
```

(2)逗号表达式可以嵌套，即一个逗号表达式又可以与另一个表达式组成一个新的表达式。例如：(x＝8＊2,x＊4),x＊2；　/＊ 整个逗号表达式的值是 32,x 的值是 16 ＊/

(3)逗号表达式还可以作为赋值运算的右边表达式来使用，例如：

x=(i=4,j=6,k=8); /* 将逗号表达式 i=4,j=6,k=8 的值赋给 x,x 的值为 8 */

(4)逗号运算符的优先级是最低的,因此下面两个表达式的作用是不同的。

① x=(z=5,5 * 2);

② x=z=5,5 * 2;

表达式①为赋值表达式,将一个逗号表达式的值赋给 x,x 的值为 10,z 的值为 5;

表达式②为逗号表达式,它包括一个赋值表达式和一个算术表达式,整个表达式的值为 10,x 和 z 的值都为 5。

(5)逗号运算符的优先级别

① 任何运算符都优于逗号运算符。

② 逗号运算符的结合性满足左结合。

2. 条件运算符和条件表达式

条件表达式是由条件运算符连接表达式构成的,其一般形式为:

e1? e2:e3;

式中“?:”为条件运算符,e1、e2、e3 是三个表达式。其中 e1 主要是关系或逻辑表达式,也可以是字符型数据或算术表达式、条件表达式、赋值表达式、逗号表达式等(值为非 0 看成逻辑真,值为 0 看成逻辑假)。e2、e3 是同类型的表达式。

表达式的含义为:当 e1 为真(非 0),表达式取 e2 的值,否则取 e3 的值。例如:

```
int a=2,b=3,c=-1,d;
d=a? b:c;      /* 由于 a 等于 2(非 0),表达式取 b 的值,d 为 3 */
a=0;
d=a? b:c;      /* 由于 a 等于 0,表达式取 c 的值,d 为-1 */
```

3. sizeof 长度运算符

sizeof 是长度运算符,属于单目运算符。sizeof 运算其一般形式为:

sizeof 表达式; 或 sizeof(类型说明符);

式中的 sizeof 分别是计算表达式或类型名的字节长度。运算结果是整型值,该值是表达式或类型名对应的数据在内存中所占的字节数。例如:

```
int a=20, b=30;
```

因为 a 是整型变量,在内存中占两个字节,所以 sizeof a 表达式的值是 2;因为 a+b 是整型表达式,所以 sizeof(a+b)表达式的值也是 2。

float 是单精度浮点型的类型名,单精度浮点型数据在内存中占 4 个字节,所以表达式 sizeof(float)的值是 4。显然,表达式 sizeof(double)的值是 8。

sizeof 运算符可以用来保证 C 语言有较好的移植性,在动态内存分配时,sizeof 也是非常有用的运算。

2.5 不同数据类型间的转换和运算

C 语言允许不同类型的数据混合使用,但需要进行数据类型的转换。

1. 表达式中的类型转换

当不同类型的常量、变量在表达式中混合运算时,C 编译系统将自动进行类型的转换工

作。它们最终将被转换为同一类型，即转换为它们之中数据位最长的类型。

转换规则如下：

(1)所有的 char、short 必定转换为 int，float 也必定转换为 double。

(2)对于所有的操作数对，运算之前总是将数据位“较短的”转换为与另一操作数相同的类型，结果为数据位“较长”的类型。

转换排列顺序为：

低　　高

int→unsigned→long→double

↑　　↑

char，short　　float

例如：

(1)一个 int 型与一个 double 型运算，先将 int 型转换为 double 型，结果为 double 型；

(2)一个 int 型与一个 long 型运算，先将 int 型转换为 long 型，结果为 long 型；

(3)两个数据中有一个为 float 型或 double 型，另一个也要转换为 double 型，结果为 double 型；

假定已指定 i 为整型变量，f 为 float 型变量，d 为 double 型变量，e 为 long 型，有下面式子：

10＋'a'＋i＊f－d/e

在计算机执行时从左至右扫描，运算次序为：

(1)先进行 i＊f 的运算，先将 i 与 f 都转换成 double 型，运算结果为 double 型；

(2)将变量 e 转换成 double 型，d/e 结果为 double 型；

(3)进行 10＋'a'的运算，先将'a'转换成整数 97，运算结果为 107；

(4)将 107 与 i＊f 的积相加，先将整数 107 转换成双精度数(小数点后加若干个 0)，结果为 double 型；

(5)将 10＋'a'＋i＊f 的结果与 d/e 的商相减，结果为 double 型。

上述的类型转换是由系统自动进行的。

2. 强制类型转换

C 还允许强制类型转换方式，可以强迫表达式的值，转换为指定类型。其一般形式为：

(类型名)表达式

```
例如：(float)a          /*将变量 a(不管它为何种类型)强制转换为单精度浮点型*/
      (int)(x+y)        /*将 x+y 的值转换成整型*/
```

强制类型转换可以将一个算术表达式或其中的某些操作数的值转换成所需要的类型，但这种转换只能将表达式的值转换成所需要的类型，并不能改变算术表达式本身的类型。此外，强制类型转换会使数据精度受到损失。

3. 计算原则

在进行不同类型表达式运算时，通常可以按照以下次序进行：

(1)如果表达式中含有 char 型、short 型，全部转换为 int 型。

(2)如果表达式中含有 float 型，全部转换成 double 型。

(3)根据表达式中运算符的优先级和结合性，确定要实施的运算以及相应的操作数。

(4)同类型的操作数运算后，结果仍为同类型；不同类型的操作数运算前，先将低精度类

型转换为高精度类型，运算结果为高精度类型。

(5)返回(3)，直到整个表达式计算完毕为止。

2.6 例题精解

【例 2.1】 字符型数据和整型数据的使用。

```
#include <stdio.h>
void main(void)
{
   char c1,c2;                          /*定义两个字符型变量 c1 和 c2*/
   c1='a';c2=98;                        /*给 c1 和 c2 赋值*/
   printf("%c   %c\n",c1,c2);           /*按字符值输出 c1 和 c2*/
   c1=c1-32;c2=c2-32;                   /*给 c1 和 c2 重新赋值*/
   printf("%c   %c\n",c1,c2);           /*按字符值输出 c1 和 c2*/
   printf("%d   %d\n",c1,c2);           /*按十进制整数值输出 c1 和 c2*/
}
```

程序运行结果，如图 2-1 所示。

图 2-1 例 2.1 程序运行结果

【解析】 程序中定义了两个字符变量 C1 和 C2，第一个 printf()函数输出 C1 和 C2 的字符值；然后经过转换，即将 C1 和 C2 转换成大写字母，由第二个 printf()函数将其转换后的大写字母输出；而第三个 printf()函数输出其对应的十进制数值。

【例 2.2】 自增运算符的使用。

```
#include <stdio.h>
void main(void)
{
   int i=1,j;                           /*定义两个整型变量 i 和 j*/
   j=i++;                               /*i 的值为 2,j 的值为 1*/
   printf("%3d%3d\n",i,j);              /*输出 i 和 j 的值*/
   j=++i;                               /*i 的值为 3,j 的值为 3*/
   printf("%3d%3d\n",i,j);              /*输出 i 和 j 的值*/
   j=(i++)+(i++);                       /*i 的值为 5,j 的值为 6*/
   printf("%3d%3d\n",i,j);              /*输出 i 和 j 的值*/
   j=(++i)+(++i);                       /*i 的值为 7,j 的值为 14*/
```

```
    printf("%3d%3d\n",i,j);                    /*输出 i 和 j 的值*/
}
```

程序运行结果，如图 2-2 所示。

图 2-2　例 2.2 程序运行结果

【解析】　此程序中使用了前置与后置自增运算符。在第一条 j=i++;语句中，i++是后置运算符，因而 i 是先取值后自增。故 j=1，i=2。第二条 j=++i;语句中，++i 是前置运算符，因而 i 是先自增，后赋值，故 j=3，i=3。第三条 j=(i++)+(i++);语句中，使用了后置运算符，故 j=6，i=5。第四条 j=(++i)+(++i);语句中，只使用了前置运算符，因而在上条语句的基础上 i 又增加了 2 次，故 j=14，i=7。

【例 2.3】　复合赋值运算符的使用。

```
#include <stdio.h>
void main(void)
{
    int i=10;
    float f=10.0;
    i+=20;
    f-=i-4;
    printf("i=%d,f=%f\n",i,f);
}
```

程序运行结果，如图 2-3 所示。

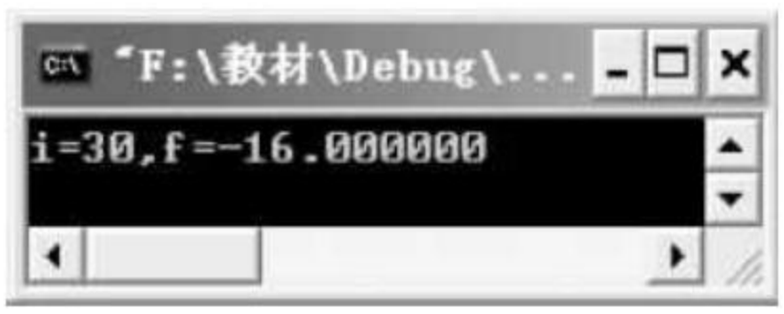


图 2-3　例 2.3 程序运行结果

【解析】　此程序是一个简单的复合赋值运算符的使用。i+=20;等价于 i=i+20;。f-=i-4;等价于 f=f-(i-4);。

2.7　本章小结

本章是 C 语言的语法基础，主要介绍 C 语言的基本数据类型、常量和变量、运算符和表达式以及常用的数学函数。

C 语言的数据类型非常丰富，其中基本类型有整型、实型、字符型和空类型。不同的数据类型，其表示方式不同、取值范围不同、占用的字节数不同、操作形式不同、输入/输出的格式也不同，因而在使用时，要掌握不同数据类型的相关要点。空类型(void)有两个用途：第一是可以用来作为函数参数，明确地表示某个函数不返回任何值(将在函数章节中介绍)；第

二是可以用“void”来定义指针类型(将在指针章节中介绍),即可定义一个指针变量,但是它不指向哪一种类型数据。

C语言的运算符包含的范围很广泛,共有34种运算符(见附录)。运算符分单目运算符、双目运算符和三目运算符。在学习过程中,要注意其操作数的个数,要掌握运算符的优先级和结合性,灵活使用各种运算符可以实现其他高级语言难以实现的运算。

C语言中由于运算符很丰富,因此表达式种类很多。常用的有算术表达式、赋值表达式、关系表达式、逻辑表达式、逗号表达式、条件表达式。在学习过程中,要熟练掌握各种表达式的类型及求值规则。表达式之间的优先次序很重要,如果表达式的优先次序不明显,可用括号指明,括号可以改变优先次序。表达式的结合顺序也能改变表达式的值。

习　　题

一、选择题

1. 下列标识符中,合法的用户标识符是_____。

A)x? 34　　B)_ab34　　C)char　　D)x@y

2. C语言中,要求参加运算的数必须是整数的运算符是_____。

A)/　　B)!　　C)%　　D)==

3. 下面运算符中,具有右结合性的是_____。

A)=　　B)+　　C)/　　D)%

4. 下列符号中,错误的转义符是_____。

A)'\"'　　B)'\\'　　C)'\81'　　D)'\x4e'

5. 字符串“abc\102\x44\\\tabcd”的长度是_____。

A)9　　B)11　　C)17　　D)19

6. 下列常量中_____是合法的?

A)2e31.65　　B)-e-3　　C)0x123　　D)"\103"

7. 下列四组常数中,正确的一组是_____。

A)160　0xffff　011　　B)-0xcdf　01a　0789

C)01　98612　0668　　D)0X37　2e5　123a

8. 下列数据中,正确的字符串常量是_____。

A)'a'　　B)"hello"

C)how do yo do　　D)'ab*cd'

9. 在C语言中,char、float和double三种类型数据所占用的内存字节数_____。

A)2,4,8　　B)1,4,8　　C)2,1,4　　D)1,1,4

10. unsigned short类型数据的取值范围是_____。

A)0～255　　B)0～65535

C)-32768～+32767　　D)0～4294967295

11. 以下合法的赋值语句是_____。

A)x=y=10　　B)x--;

C)x+y;　　D)c=int(a+b);

12. 若已定义 x 和 y 为 double 型变量，则表达式 x=1，y=x+3/2 的值是_____。

A)1　　B)2　　C)2.0　　D)2.5

13. 当用 #define A 18.5 定义后，下列叙述正确的是_____。

A)A 是符号常量　　B)A 是实型变量

C)A 是一个字符　　D)A 是字符串常数

14. 设有语句 int a=4;，则执行了语句 a+=a-=a*a 后，变量 a 的值是_____。

A)24　　B)-24　　C)4　　D)16

15. 设整型变量 x，y，z 均为 3，表达式 x+++y+++z++ 的值是_____。

A)9　　B)12　　C)13　　D)15

16. 下面表达式中符合 C 语言语法的赋值表达式是_____。

A)a=5+c+d=a+5　　B)a=c+d++=a+5

C)a=(5+b,d++,a+5)　　D)a=5+c,d=a+5

17. 已知 int i;float d;，正确的语句是_____。

A)(int d)　　B)int(d)%i　　C)int(d%i)　　D)(int)d%i

二、填空题

1. 设有变量定义 char a='\72';则变量 a 包含了_____个字符。

2. 在 C 语言中，char 型数据在内存中是以_____形式存储的。

3. 已知 i=5，执行语句 k=(i++)+(i++)+(i++)后，k 的值为_____，i 的值为_____。

4. 设有 int x=11;则表达式(x++*1/3)的值为_____。

5. 下列程序段执行后，a、b、c 和 d 的值分别是_____。

```
int a=2;
int b=5;
int c=a++;
a--=2;
d=(++a,a+b);
```

6. 求下面算术表达式的值。

(1)x+a%3*(int)(x+y)%2/4

设 x=2.5，a=8，y=4.8

(2)(float)(a+b)/2+(int)x%(int)y

设 a=3，b=4，x=5.5，y=2.5

(1)表达式的值是_____；

(2)表达式的值是_____。

7. 设有下面语句

```
int x,y,z;
x=y=z=4;
```

(1)执行“x--=y-z”后 x=_____；

(2)执行“x%=y+z”后 x=_____。

第3章 顺序结构程序设计

【教学提示】

结构化程序设计的基本思想是:任何程序都可以采用三种基本结构来构造,这三种基本结构是:顺序结构、分支结构和循环结构。

顺序结构是最简单也是最基本的程序结构,它按照语句出现的先后顺序依次执行程序。顺序结构主要由简单语句、复合语句及输入输出语句构成。

【核心概念】

表达式的求值规则 字符输入(getchar())/输出(putchar())函数 格式输入(scanf())/输出(printf())函数

3.1 C语言中的语句

C语句大致可分为以下5类:

(1)表达式语句;

(2)函数调用语句;

(3)控制语句;

(4)复合语句;

(5)空语句。

3.1.1 表达式语句

在任何一个表达式的后面加一个分号就构成了一条表达式语句。

例如:

```
a+b      /* 表达式 */
a+b;     /* 表达式语句 */
```

在C语言中,赋值和函数调用都是表达式,所以赋值语句和函数调用语句也是一种特殊的表达式语句。

一般形式为:

表达式;

执行表达式语句就是计算表达式的值。

例如:赋值表达式 x=3,在此表达式后加一个分号(即 x=3;)就构成一条赋值语句。

注意:赋值符号(=)的左边只能是变量,而在“=”的右边可以是变量、常量及各种表达式等。在实际应用中,要注意表达式和表达式语句的区别。

3.1.2 复合语句

复合语句是用大括号括起来的若干语句,这些语句被看成一个整体。在程序中应把复

合语句看成是单条语句，而不是多条语句。

复合语句的一般形式为：

```
{
   语句说明;
   可执行语句;
}
```

例如：

```
{
   int x,a;
   x=y+z;
   a=b+c;
   printf("%d%d",x,a);
}
```

是一条复合语句。

注意：复合语句在语法上相当于一个语句，凡单一语句可以存在的位置都可以使用复合语句。如果复合语句中只有一个语句，大括号可以省略。

3.1.3　控制语句

控制语句用于控制程序的流程，以实现程序的各种结构方式。它们由特定的语句定义符组成。C 语言有 9 种控制语句，可分成以下三类：

条件判断语句：if 语句、switch 语句；

循环执行语句：do while 语句、while 语句、for 语句；

转向语句：break 语句、goto 语句、continue 语句、return 语句。

3.1.4　函数调用语句

由一个函数调用加一个分号构成函数调用语句。其一般形式为：

函数名(实际参数表);

执行函数语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调函数体中的语句，求取函数值(具体函数如何调用，参数如何传递详见第 8 章)。简单的函数调用语句，例如：

```
printf("How do you do?");      /* 调用库函数,输出字符串 */
```

3.1.5　空语句

只有分号“;”组成的语句称为空语句，空语句是什么也不执行。在程序中空语句可用来作空循环体。例如：

```
while(getchar()!='\n')
;      /* 空语句 */
```

本语句的功能是：不断从键盘输入字符，只要输入的字符不是回车则重新输入。这里的

循环体为空语句。

空语句也可以看作是一种特殊情况下的表达式语句,它只有一个分号,执行时不做任何事情。

3.2 数据的输入输出

完整的程序都应含有数据的输入/输出功能。没有输出功能的程序是无用的,因为运算结果看不见;没有输入功能的程序,由于只能对固有的数据加工,所以是单调、死板的。因此,输入/输出功能对每一个程序都是不可缺少的。

3.2.1 数据输入输出概念

数据的输入输出是对存贮数据的计算机主机而言,计算机通过显示器、打印机等外设将数据显示、打印或存放在磁盘上称之为“输出”。而通过键盘、扫描仪、磁盘、光盘把数据送入计算机内部称之为“输入”。每一种高级语言都有其灵活的输入输出功能。

一般的高级语言的输入输出是通过输入输出语句实现的,而C语言本身没有提供输入输出语句,它的输入输出功能是通过调用编译系统提供的输入输出库函数来实现的。由于使用库函数作为输入输出手段,避免了系统在编译阶段处理与硬件有关的问题,不仅简化了编译系统,而且通用性强,可移植性好,可在各种型号的计算机上使用。

C语言编译系统提供的标准库函数存放在不同的头文件(也称标题文件)中。例如,格式输入输出函数scanf()和printf(),字符输入输出函数getchar()和putchar()等存放在标准输入输出头文件stdio.h中。而sin(x),cos(x),sqrt(x)等数学函数存放在标准输入输出头文件math.h中。这些头文件中存放了关于这些函数的说明、类型和宏定义,而对应的子程序则存放在运行库“.lib”中。使用时必须在程序的开头用预编译命令“#include”将头文件包含到用户程序中去。其一般形式为:

#include<头文件>或#include“头文件”

3.2.2 字符数据的输入输出

输入函数getchar()和输出函数putchar()是单个字符输入输出函数。

1.字符输入函数getchar()

getchar()函数的功能是从标准输入设备(键盘)读入一个字符。getchar()函数没有参数,因此本身不能提供有效存放所读字符的变量,其一般调用形式为:

c=getchar();

表示从键盘输入一个字符赋给对应的字符变量c。

使用getchar()函数时需注意:

如果getchar()函数读入的字符不赋给任何变量,则该函数只能作为表达式的一部分使用。

注意:getchar函数只能接受单个字符,输入数字也按字符处理。输入多于一个字符时,只接收第一个字符。

2.字符输出函数putchar()

putchar()函数的功能是向标准输出设备(显示器)输出一个字符。putchar()为带参数

函数，其一般调用形式为：

putchar(c);

其中，c 为一整型或字符型变量，也可以是字符或整型常量。

当 c 为字符型常量时，输出的是该字符；当 c 为整型常量时，输出的是该常量值对应的 ASCII 字符。例如：

```
putchar('A');        /* 输出 A 字符 */
putchar(65);         /* 输出 ASCII 码值为 65 的字符，也是 A */
```

转义字符也可作为字符常量输出，例如：

```
putchar('\n');       /* 输出一个换行符 */
```

注意：当程序中使用了 getchar()/putchar()函数，则该程序的开始处要加上 #include <stdio.h>头文件。

【例 3.1】　阅读下面程序，写出程序结果。

```
#include <stdio.h>                    /* getchar()/putchar()原型包含文件 */
void main(void)
{
   int a1,a2;                         /* 定义两个整型变量 */
   char c1,c2;                        /* 定义两个字符变量 */
   a1=97;a2=65;
   c1='a';c2='A';
   putchar(a1);putchar(a2);
   putchar(c1);putchar(c2);
   putchar('\n');                     /* 输出字符及换行符 */
   printf("请输入字符:");
   a1=getchar();
   c1=getchar();                      /* 从键盘输入字符 */
   putchar(a1);
   putchar('\t');
   putchar(c1);                       /* 输出字符及制表符 */
}
```

程序运行结果，如图 3-1 所示。

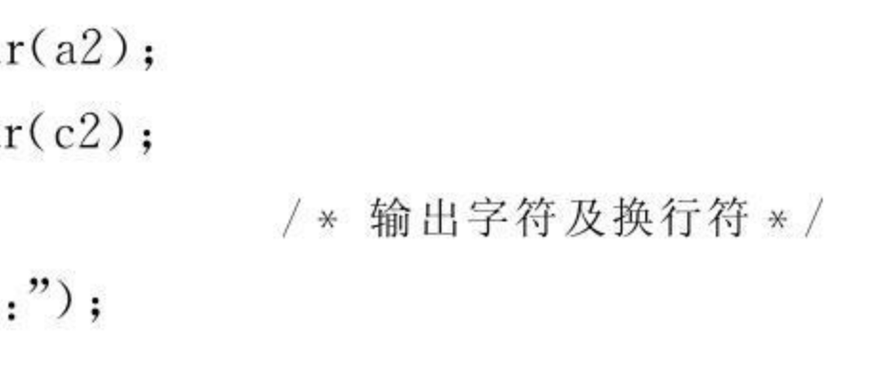


图 3-1　例 3.1 程序运行结果

注意：在键盘输入字符时，字符与字符之间不能加分隔符。

3.2.3 格式输入与输出

格式输入函数 scanf()和格式输出函数 printf()功能更强,使用更灵活,它们的函数原型在头文件“stdio.h”中。作为特例,有的系统不要求在使用这两个函数前必须包含 stdio.h 文件。而对于 VC++6.0 运行环境,程序中必须加上“stdio.h”包含文件。

1.格式输出函数 printf()

(1)printf()函数调用形式

其一般形式为:

printf("格式控制字符串",输出表列);

这里的“格式控制字符串”和输出表列可以看作是该函数的参数。它的功能是按指定的输出格式向显示屏幕输出信息。

其中,格式控制字符串用于指定输出格式,它有三种形式:

① 格式说明符:用来规定相应输出表列内容的输出格式,例如:%d、%f。

② 转义字符:用来输出转义字符所代表的控制代码或特殊字符,常用的有:回车换行“\n”,水平制表“\t”。

③ 普通字符,需要原样输出的字符,例"input a:"、"a="。

输出表列:需要输出的若干个数据项,它与格式说明符在数量和类型上应该一一对应。例如:

```
#include  <stdio.h>
void main(void)
{
   int a=10,b=15;
   printf("a=%d,b=%d,a+b=%d\n",a,b,a+b);
}
```

程序运行结果:a=10,b=15,a+b=25。

(2)格式说明符

格式说明符的表示方法:以百分号“%”打头,后跟一个小写英文字母,两者中间还可以以 m.nL 格式插入附加格式符,以使输出格式更为丰富灵活,格式说明符和附加格式说明符以及格式输出字符表的说明及其含义,见表 3-1、表 3-2 和表 3-3 所列。

表 3-1 printf()函数格式说明符表

格式说明符	说　　明
d	以十进制形式输出带符号整数(正数省略符号)
o	以八进制形式输出无符号整数(不输出前导符 0)
x	以十六进制形式输出无符号整数(不输出前导符 0x)
u	以十进制进制形式输出无符号整数
f	以小数形式输出单、双精度实数,隐含输出 6 位小数
e	以指数形式输出单、双精度实数,数字部分小数位数为 6
g	选用%f 或%e 中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

表 3-2　printf()函数附加格式说明符表

附加格式说明符	说　　明
字母 l	用于长整型数据,可加在 d,o,x,u 格式符前
m(为一正整数)	指定输出数据所占宽度(含小数点)
n(为一正整数)	对实数,表示输出 n 位小数,对字符串,表示截取的字符个数
+(通常省略)	右对齐,即输出的数字或字符在域内向右靠,左边填空格
-	左对齐,即输出的数字或字符在域内向左靠,右边填空格

表 3-3　格式输出字符表

格式字符	数据对象	输出形式	数据输出方法
%-md	int、float unsigned int unsigned short char	十进制整数	无 m 按实际位数输出。有 m 输出 m 位; 超过 m 位,按实际位数输出;不足 m 位补空格。 无+右对齐(左补空格) 有-左对齐(右补空格)
%-mo		八进制整数	
%-mx		十六进制整数	
%-mu		无符号整数	
%-mld	long unsigned long	十进制整数	无 m 按实际位数输出。有 m 输出 m 位;超过 m 位,按实际位数输出;不足 m 位补空格。 无+右对齐(左补空格)。 有-左对齐(右补空格)
%-mlo		八进制整数	
%-mlx		十六进制整数	
%-mlu		无符号整数	
%-m.nf	float double	十进制小数	无 m 按实际位数输出。有 m 输出 m 位; 超过 m 位,按实际位数输出;不足 m 位补空格。 无+右对齐(左补空格) 有-左对齐(右补空格)
%-m.ne		八进制指数	
%-g		自动选取 f 或 e 中宽度的格式	
%-mc	char int short	单个字符	无 m 输出单个字符,有 m 输出 m 位,不足宽度补空格。 无+右对齐(左补空格) 有-左对齐(右补空格)
%-m.ns	字符串	一串字符	无 m.n 按实际字符串输出全部字符; 有 m.n 仅输出前 n 个字符,补空格。 无-右对齐(左补空格) 有-左对齐(右补空格)

下面我们以两个实例来总结一下 printf()函数的使用方法。

【例 3.2】　格式输出函数的用法。

```
#include <stdio.h>
void main(void)
{
   int a=12;
   float b=123.12345678;
   double c=12345678.1234567;
   char d='p';
```

```
    printf("a=%d,%5d,%o,%x\n",a,a,a,a);
    printf("b=%f,%lf,%5.4lf,%e\n",b,b,b,b);
    printf("c=%lf;%f,%8.4lf\n",c,c,c);
    printf("d=%c,%6c\n",d,d);
}
```

程序运行结果,如图 3-2 所示。

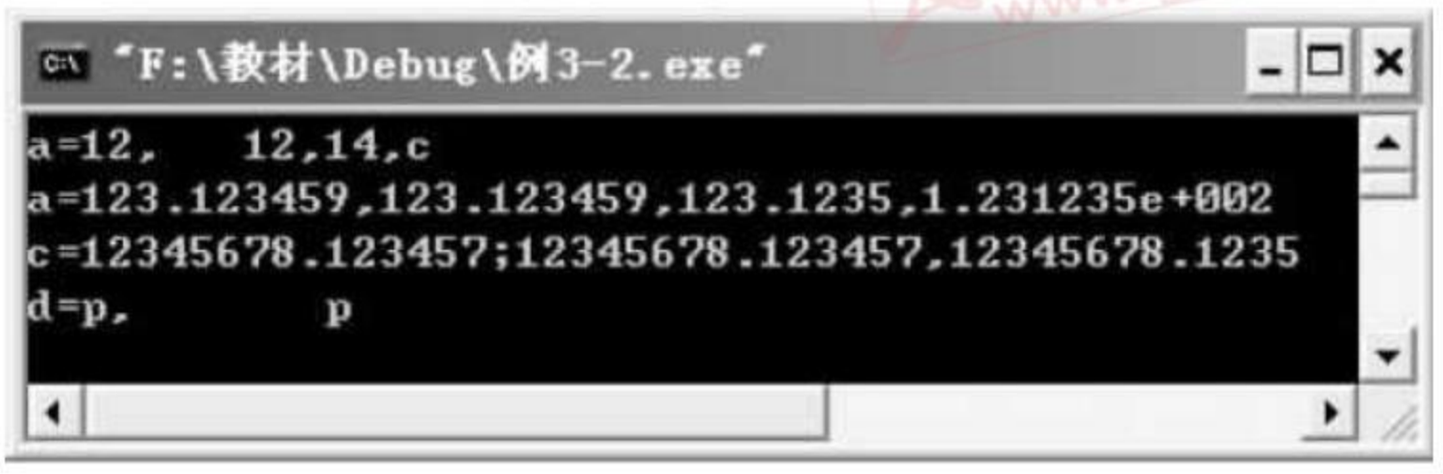


图 3-2　例 3.2 程序运行结果

【解析】 在上例中,对%f 不指定字符宽度,系统自定为全部整数位加 6 位小数,并且 l 对 f 格式无影响,如第 2 个 printf()中的%f 和%lf 输出格式相同;当用 m 指定宽度输出时,若数据位数小于 m,则左端补空格,如第一个 printf()中的%5d,若大于 m,则按实际位输出,如第二个 printf()中的%5.4lf;当用 .n 指定小数位数时,多余部分被截去,如第二个 printf()中的%5.4lf 和第 4 个 printf()中的%8.4lf。

【例 3.3】 阅读下面程序,写出程序运行结果。

```
#include<stdio.h>
void main(void)
{
    long b=-1;
    float f=123.456;
    char c='a';
    printf("b=%ld,b=%lo,b=%lx,b=%lu\n",b,b,b,b);
    printf("f=%f,f=%7.2f,f=%-7.2f\nf=%e,f=%g\n",f,f,f,f,f);
    printf("c=%c,c=%3c,c=%-3c,c=%d,c=%c\n",c,c,c,'a',65);
    printf("s1=%s,s2=%7.3s,s3=%-7.3s\n","12345","ABCD","12345");
}
```

程序运行结果,如图 3-3 所示。

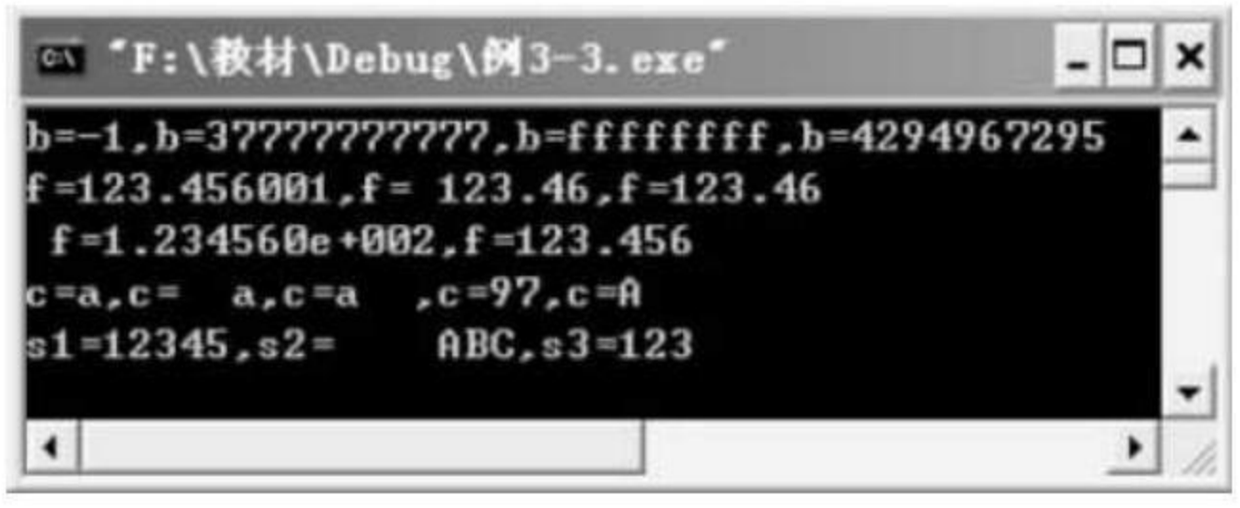


图 3-3　例 3.3 程序运行结果

【解析】 在上例中，对长整型数，由于内存占 4 个字节，其取值范围不同进制数取值不同。printf()中用逗号作格式字符的分隔符，则运行结果亦是逗号分隔，如无分隔符，输出结果也无分隔符，如程序中的第 4 个 printf()函数；%s 用来输出一串字符；%c 的作用同 putchar()函数，仅输出一个字符。

一个整数，只要它的值在 0～255 范围内，可以用字符形式输出，其输出值为该数对应 ASCII 字符，如第 4 个 printf()中的%3c 和%－3c，反之字符也可用整数形式输出，输出的将是这个字符的对应 ASCII 值，如第 2 个 printf()中的%d；当用－m 指定宽度时，数据位数小于 m，则右端补空格，如第 2 个 printf()中的%－7.2f 和第 3 个 printf()中的%－3c。

其他一些用法读者可以自己分析。

2. 格式输入函数 scanf()

(1)一般形式

scanf("格式控制字符串",地址表列);

这里的“格式控制字符串”和“地址表列”也可以看作是函数的参数。它的功能是按指定的输入格式从键盘接受用户输入的信息。

其中：格式控制字符串与 printf()函数中的含义相似，所不同的是它是对输入格式进行控制，它的内容可以是格式说明符或普通字符，转义字符则较少使用。

地址表列：由若干个等待输入的数据所对应的内存单元地址组成。

地址表之间用逗号分隔，在 C 语言中，地址量的表示是在变量前加前缀符号“&”，如“&a”表示变量 a 的地址，地址表列在数量和类型上也应与“格式控制字符串”中的格式说明符一一对应。

【例 3.4】 用 scanf()函数输入数据。

```
#include<stdio.h>
void main(void)
{
   int a,b,c;
   scanf("%d%d%d",&a,&b,&c);
   printf("%d,%d,%d\n",a,b,c);
}
```

程序运行结果，如图 3－4 所示。

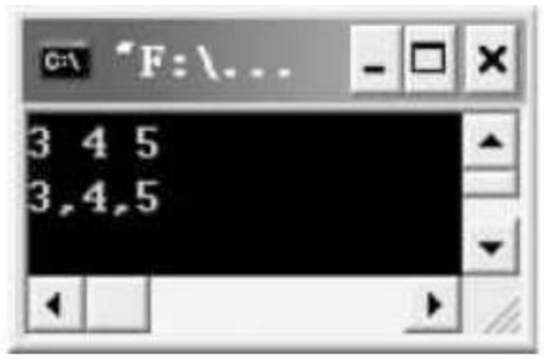


图 3－4　例 3.4 程序运行结果

【解析】 “%d%d%d”表示按十进制整数格式输入数据，由于中间无分隔号，故在输入数据时要用一个或一个以上的空格或回车键作为数据间的间隔。

(2)格式说明符

scanf()函数可使用的格式说明符与 printf()函数中介绍的基本相同,只有少数几个不适合作为输入格式说明符使用的例外,见表 3-4、表 3-5 所列。

表 3-4 scanf()函数格式说明符

格式说明符	说　　明
d	输入十进制整数
o	输入八进制整数
x	输入十六进制整数
f	输入实数,以小数形式输入
e	输入实数,以指数形式输入
c	输入单个字符
s	输入字符串

表 3-5 scanf()函数附加格式说明符表

附加格式说明符	说　　明
字母 l	输入长整型数据(%ld,%lo,%lx)及 double 型数据(%lf 或%le)
h	输入短型整数据(%hd,%ho,%hx)
m(为一正整数)	指定输入数据所占的宽度,域宽应为正整数
*	对应的输入项读入后不赋给相应的变量

说明:

① 对 unsigned 型变量所需的数据,可以用%u,%d 或%o,%x 格式输入。

②“*”符号表示该输入项读入后不赋给相应的变量,即跳过该输入值。例如:

```
scanf("%d%*d%d",&a,&b);
```

当输入 1 2 3 时,系统将 1 赋给 a,2 被跳过,3 赋给 b。

(3)使用 scanf()函数应注意:

① 输入数据时不能规定精度,如 scanf("%7.2f",&a);是不合法的,不能企图用该语句输入小数为 2 位的实数。

② scanf()函数要求给出的是变量地址,而不是变量名,这是 C 语言与其他高级语言的不同之处,初学者应特别注意,例 scanf("%d",a);是非法的,变量名不能出现在地址表列中。

③ 输入多个数据时,数据输入时的分隔符应与“格式说明符”中的分隔符相对应。

假设 a 的值为 3,b 的值为 4。例如:

```
scanf("%d,%d",&a,&b);              /*应输入 3,4*/
scanf("a=%d,b=%d",&a,&b);          /*应输入 a=3,b=4 形式*/
```

④若格式说明符中无分隔符,可用空格、TAB 或回车符作数据的分隔符。C 编译在遇到空格、TAB、回车符或非法数据(如对"%d"格式输入"12A"时,A 即为非法数据)时,即认为该数据结束。例如:

```
scanf("%d%d",&a,&b);               /*应输入 3  4*/
```

⑤在用%c 格式输入单个字符时，若格式说明符串中没有分隔符，则认为所有输入的字符均为有效字符。例如：

```
scanf("%c%c%c",&a,&b,&c);
```

若输入：3　4　5，则把 3 送入 a，空格送入 b，4 送入 c，因为系统把空格也作为一个字符输入，对应送给 b。

同③一样，如果格式说明符中加入了分隔符，则输入时可加入相应分隔符。例如：

```
scanf("%c?%c",&a,&b);          /*可输入 3? 4*/
scanf("%c,%c",&a,&b);          /*可输入 3,4*/
```

从以上分析可以看出字符与格式输入输出函数的异同点：

共同点：格式输入输出函数 scanf()和 printf()与字符输入输出函数 getchar()和 putchar()均通过系统提供的输入输出标准函数来实现的。scanf()和 getchar()都是接收来自键盘的输入数据，printf()和 putchar()都是向显示器屏幕输出数据，它们均包含在头文件stdio. h 中。

不同点：功能上格式输入输出函数功能较强，对输入或输出的信息几乎没有限制，而字符输入输出函数功能较单一，每次只能输入输出一个字符，两者相比，前者能实现后者的功能；从使用方法上看，格式输入输出函数使用灵活，合理而巧妙地使用各种格式说明符和提示信息，可使屏幕丰富多彩，而字符输入输出函数则不具备这些灵活性，但是 putchar()、getchar()比较简单、明了，实际应用时应考虑它们各自的特点，合理使用它们。

格式输入字符及所控制的数据类型、形式和方法，见表 3-6 所列。

表 3-6　格式输入字符表

格式字符	数据对象	输入形式	数据输入方法
%md	int、short、unsigned int、unsigned short	十进制整数	无 m 按实际位数输入；有 m 输入 m 位，不足 m 则跟回车键。
%mo		八进制整数	
%mx		十六进制整数	
%mld	long unsigned long	十进制整数	无 m 按实际位数输入；有 m 输入 m 位，不足 m 则跟回车键。
%mlo		八进制整数	
%mlx		十六进制整数	
%mlf	float double	十进制整数	无 m 按实际位数输入；有 m 输入 m 位，不足 m 则跟回车键。
%mle		十进制整数	
%mle	char	单个字符	无 m 仅取单个字符；有 m 输入 m 位，仅取第一个字符。
%ms	字符串	一串字符	无 m 取回车或空格前若干个字符；有 m 仅取前 m 字符。

3.3　例题精解

【例 3.5】　阅读程序，写出运行结果。

```
#include <stdio.h>
void main(void)
```

```
{
   char c1,c2;
   int n1,n2;
   c1=getchar();
   c2=getchar();
   n1=c1-'0';
   n2=n1*10+(c2-'0');
   printf("%d,%d,%d\n",c1,c2,n2);
}
```

程序运行时输入:23 ↙,则输出结果如图 3-5 所示。

图 3-5 例 3.5 程序运行结果

【解析】 从键盘输入 23 ↙ 时,字符 2 被第一个 getchar()接收赋给 c1,字符 3 被第二个 getchar()接收赋给 c2,通过 c1-'0'运行后,变量 n1 得到整数值 2,再经过 n1*10+(c2-'0')运算后,变量 n2 得到整数值 23,因此程序的输出结果为十进制数的:23。

【例 3.6】 输入三角形的三边长,求三角形面积。

已知三角形的三边长 a、b、c,则该三角形的面积公式为:

$area=\sqrt{s(s-a)(s-b)(s-c)}$,其中 s=(a+b+c)/2

```
#include <stdio.h>
#include <math.h>
void main(void)
{
   int a,b,c;double s,area;
   scanf("%d,%d,%d",&a,&b,&c);
   s=(a+b+c)/2.0;
   area=sqrt(s*(s-a)*(s-b)*(s-c));
   printf("a=%d,b=%d,c=%d,s=%7.2f\n",a,b,c,s);
   printf("area=%f\n",area);
}
```

程序运行结果,如图 3-6 所示。

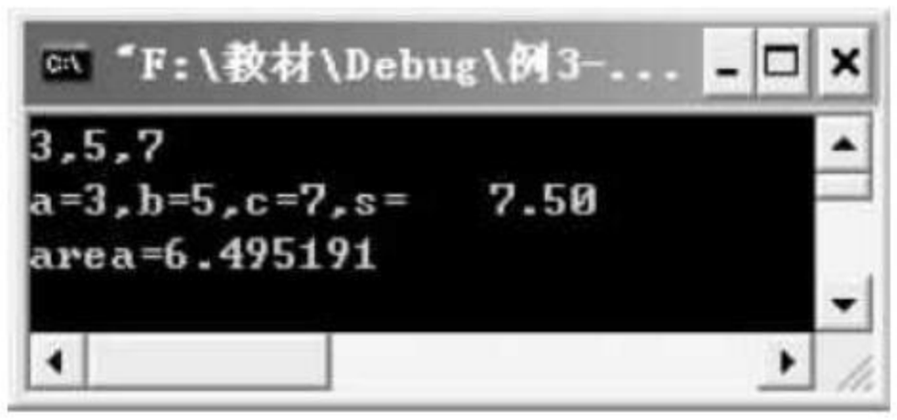


图 3-6 例 3.6 程序运行结果

【解析】 程序的编写有三部分组成:定义、输入部分;计算处理部分;输出部分。该程序中,由于 a、b、c 是未知量,因而通过键盘输入; s 和 area 是要求解的表达式,可以根据已知公式进行计算;最后将求出的结果,按照给定格式输出。

在程序的第 7 行中 sqrt()是求平方根的函数。此数字函数的原型及有关信息,包含在"math.h"库文件中。因而在调用前,将被包含文件"include<math.h>"嵌入到源文件中。

【例 3.7】 求 $ax^2+bx+c=0$ 方程的根,a、b、c 由键盘输入,设 $b^2-4ac>0$。求根公

式为：

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}\quad 令：p=\frac{-b}{2a},q=\frac{\sqrt{b^2-4ac}}{2a}$$

则　$x_1=p+q,x_2=p-q$

```
#include <stdio.h>
#include <math.h>
void main(void)
{
   int a,b,c;
   double disc,x1,x2,p,q;
   scanf("%d,%d,%d",&a,&b,&c);
   disc=b*b-4*a*c;
   p=-b/(2.0*a);
   q=sqrt(disc)/(2.0*a);
   x1=p+q;x2=p-q;
   printf("\nx1=%7.3f\nx2=%7.3f\n",x1,x2);
}
```

程序运行结果，如图 3-7 所示。

图 3-7　例 3.7 程序运行结果

【解析】　一元二次方程的根有多种，这是仅求两个实根。根据题意，a、b、c 由键盘输入，计算出两个实根。由于公式中要求平方根，需借助于平方根函数，因而包含文件 #include <math.h> 千万不能丢。

【例 3.8】　从键盘输入一个小写字母，将其转换成大写字母输出。

```
#include <stdio.h>
void main(void)
{
   char c;
   c=getchar();
   printf("%c,%d\n",c,c);
   c=c-32;
   printf("%c,%d\n",c,c);
}
```

程序运行结果，如图 3-8 所示。

图 3-8　例 3.8 程序运行结果

【解析】　该程序通过 getchar()函数从键盘输入一个小写字母，然后经过转换，即该字符减去 32，得到一个大写字母。

3.4 本章小结

顺序结构是C语言三种控制结构中最简单的一种。常见的顺序结构程序设计的语句有:声明语句、赋值语句和输入输出函数。

本章主要阐述了在程序设计中不可缺少的输入/输出函数。格式输入/输出函数(scanf()/printf())用于对各种数据的输入/输出,在使用时要注意不同类型格式描述符的作用,注意scanf()函数输入格式的控制,以及printf()函数输出格式的控制。字符输入/输出函数(getchar()/putchar())是专用于对字符数据的输入/输出,getchar()是从标准输入设备读入一个字符,putchar()是通过标准输出设备输出一个字符。输入和输出函数包含在C语言的标准函数库中,使用时必须在程序的开始处加上"#include <stdio.h>"文件包含预处理命令。

习 题

一、单项选择题

1. 若用scanf()输入长整型数据,则错误的格式说明符是_____。

A)%d B)%ld C)%lo D)%lx

2. 若用printf()输出双精度型数据,则错误的格式说明符是_____。

A)%-10.2f B)%g C)%e D)%10.2lf

3. putchar函数可以向终端输出一个_____。

A)整型变量表达式的值 B)实型变量值

C)字符串 D)字符或字符变量值

4. 若有语句scanf("%d%c%d",&a,&b,&c);假设输入序列为123a345o123,则a,b,c的值为_____。

A)无值 B)123,a,345o.12 C)123,a,无 D)123,a,345

5. 执行下列程序

```
#include <stdio.h>
void main(void)
{
  int y=3,x=3,z=1;
  printf("%d,%d\n",(++x,y++),z+2);
}
```

后输出结果是_____。

A)3,4 B)4,2 C)4,3 D)3,3

6. 若有语句scanf("%d%c%d%c",&a,&x,&b,&y);要使变量a,b分别得到23,45;使x,y分别得到A,B;则错误的输入形式为_____。

A)23A 45B B)23 A 45 B C)23A45B D)23A ↙
45B ↙

7. 若有以下定义和语句：

```
int u=010,v=0x10,w=10;
printf("%d,%d,%d",u,v,w);
```

则输出结果是_____。

A)8,16,10　　B)10,10,10　　C)8,8,10　　D)8,10,10

8. 若 x 是 unsigned short 型变量，则执行下列语句后，x 值为_____。

```
x=65535;
printf("%d",x);
```

A)65535　　B)1　　C)-1　　D)无定值

9. 以下程序的运行结果是_____。

```
#include <stdio.h>
void main()
{
   printf("%d\t",NULL);
}
```

A)1　　B)0　　C)-1　　D)不确定值

10. 设 x、y 均为 int 型变量，且 x=8,y=4，则执行下面语句

```
printf("%d,%d\n",x++,--y);
```

则输出结果是_____。

A)8,4　　B)8,3　　C)9,3　　D)9,4

11. 以下程序

```
#include <stdio.h>
void main(void)
{
   int a,b=1,c=246;
   a=c/100%9;
   b=b++;
   printf("%d,%d\n",a,b);
}
```

的输出结果是_____。

A)2,2　　B)3,2　　C)4,3　　D)2,-1

12. 若 d1,d2,d3,d4 均为 char 型变量，则下面语句执行后结果为_____。

```
d1='1';d2='2';
d3='3';d4='4';
printf("%1c\n",d1);
printf("%2c\n",d2);
printf("%3c\n",d3);
printf("%4c\n",d4);
```

A)1　　B)1　　C)1　　D)输出格式说明符不合法

2　　2　　02

3　　3　　003

4　　4　　0004

二、填空题

1. C语言的表达式后加一个_____就构成表达式语句。

2. 字符输出函数 putchar()用于输出_____，使用该函数时，需要在程序头部写入_____语句。

3. 标准格式输出函数 printf()的功能是_____，其中“%”号是_____。

4. 格式说明符是用来_____；“o”字符表示_____；“c”字符表示_____。

5. 设有定义语句

```
double t=3,t1;
t1=(t,t+5,++t);
printf("%e,%f\n",t1,t);
```

输出是_____。

6. 以下程序的输出结果为_____。

```
#include <stdio.h>
void main(void)
{
   printf("%f,%4.3f\n",3.14,3.1415);
}
```

7. 若有如下定义：

```
int a;float x;char c;
```

假设要使 a=3,x=12.6,c='A',正确的 scanf 函数输入格式是_____。

8. 下面程序的输出结果为 16.00，请将程序补充完整。

```
#include <stdio.h>
void main(void)
{
   int a=9,b=2;
   float x=6.6,y=1.1,z;
   z=a/2+b*x/y+1/2;
   printf("_____",z);
}
```

三、应用题

1. 下面程序的运行结果为_____。

```
#include <stdio.h>
void main(void)
{
   char c1='a',c2='b',c3='c',c4='\101',c5='\116';
   printf("a%cb%c\tc%c\tabc\n",c1,c2,c3);
```

```
    printf("\t\b%c%c\n",c4,c5);
}
```

2. 下面程序的运行结果为_____。

```
#include <stdio.h>
void main(void)
{
    int c1,c2;
    c1=97;
    c2=98;
    printf("%c%c\n",c1,c2);
    printf("%d%d\n",c1,c2);
}
```

3. 下面程序的运行结果为_____。

```
#include <stdio.h>
void main(void)
{
    int i,j,m,n;
    i=6;
    j=8;
    m=++i;
    n=j++;
    printf("%d,%d,%d,%d",i,j,m,n);
}
```

4. 下面程序的运行结果为_____。

```
#include <stdio.h>
void main(void)
{
    int a=1,b=2,c=3;
    a=b;
    b=c;
    c=a;
    printf("a=%d,b=%d,c=%d\n",a,b,c);
}
```

5. 有下面程序，运行结果是_____。

```
#include <stdio.h>
void main(void)
{
    int a,b,c;
    a=(b=(c=2)*5)*3-4;
```

```
    printf("a=%d;b=%d;c=%d\n",a,b,c);
}
```

6. 有下面程序,运行结果是_____。

```
#include <stdio.h>
void main(void)
{
    int a=4,b=8;
    float x=12.74325;
    char c='B';
    printf("%3d%3d\n",a,b);
    printf("%d,%f\n",a,x);
    printf("%7.2f,%9.4f\n",x,x);
    printf("%c,%d",c,c);
    printf("%s,%10s\n","computer","computer");
}
```

四、编写程序题

1. 编写程序,用 getchar()函数读入两个字符给 c1、c2,然后分别用 putchar()函数和 printf()函数输出这两个字符。
2. 按下列公式计算并输出 S 的值,其中 a 和 b 的值由键盘输入。

$$S=\frac{2ab}{(a+b)^2}$$

3. 编写程序,从键盘输入一个大写字母,将该字母转换成小写字母。

第 4 章　选择结构程序设计

【教学提示】

设计选择结构程序，要考虑两个方面的问题：一是在 C 语言中如何表示条件，二是在 C 语言中实现选择结构用什么语句。在 C 语言中表示条件，一般用关系表达式或逻辑表达式，实现选择结构用 if 语句或 switch 语句。本章要求熟练掌握 if 和 switch 语句的语法及使用，掌握利用分支控制语句编写选择结构程序的方法。

【核心概念】

关系运算　逻辑运算　if 语句　条件运算符　switch 语句

4.1　关系运算和逻辑运算

要理解关系运算和逻辑运算的概念，就必须了解真与假的表示方法。在 C 语言中，任何非 0 值都可以表示真，0 值表示假。如果关系或逻辑运算结果为非 0，则表达式值为 1，否则表达式值为 0。

4.1.1　关系运算

所谓“关系运算”实际上就是“比较运算”，即将两个数据进行比较，判定两个数据是否符合给定的关系。

例如，“a＞b”中的“＞”表示一个大于关系运算。如果 a 的值是 5，b 的值是 3，则大于关系运算“＞”的结果为“真”，即条件成立；如果 a 的值是 2，b 的值是 3，则大于关系运算“＞”的结果为“假”，即条件不成立。在 C 语言中，“真”值用 1 表示，“假”值用 0 表示。

1. 关系运算符

C 语言提供 6 种关系运算符：

＜(小于)　　＜＝(小于或等于)　　＞(大于)

＞＝(大于或等于)　　＝＝(等于)　　!＝(不等于)

注意：在 C 语言中，“等于”关系运算符是双等号“＝＝”，而不是单等号“＝”(赋值运算符)。

2. 优先级

(1)在关系运算符中，其中＜，＜＝，＞，＞＝的优先级相同，＝＝，!＝优先级相同，前四种优先级高于后两种。

(2)与其他种类运算符的优先级关系

注意：关系运算符的优先级，低于算术运算符，但高于赋值运算符。

3. 关系表达式

用关系运算符组成的表达式称为关系表达式。

例如：

```
3 * 5>1+8          /* 即 15>9,关系成立,结果为真,表达式的值为 1 */
3+4!=2+5           /* 即 7!=7,关系不成立,结果为假,表达式的值为 0 */
```

因为关系运算的结果可为 0 或 1，所以，可以参与运算。例如：

```
3+(3<1+5)
```

括号中的关系表达式值为 1，整个表达式的值为 4。

在关系表达式运算时，千万不要将赋值表达式与关系表达式混淆，比如，x=5 是赋值表达式，其值为 5；x==5 是关系表达式，如果 x 为 5 则为“真”，取值为 1；否则为“假”，取值为 0。

难点：由于关系运算符的结合性从左到右，所以关系表达式 5<x<20 等价于(5<x)<20；又因为 5<x 的结果只能为 1 或 0，于是，无论 x 取任何值，5<x<20 的值都是“真”，这同时说明关系表达式 5<x<20 并不等同于数学式 5<x<20。

4.1.2 逻辑运算

关系表达式只能描述单一条件，例如“x>=0”。如果需要描述“x>=0”同时“x<10”，就要借助于逻辑表达式了。

1. 逻辑运算符及其运算规则

(1)C 语言提供三种逻辑运算符

```
&&      /* 逻辑与(相当于“同时”) */
||      /* 逻辑或(相当于“或者”) */
!       /* 逻辑非(相当于“否定”) */
```

例如，下面的表达式都是逻辑表达式：

```
(x>=0) && (x<10)
(x<1) || (x>5)
!(x==0)
(year%4==0)&&(year%100!=0)||(year%400==0)
```

(2)运算规则

① && 当且仅当两个运算量的值都为“真”时，运算结果为“真”，否则为“假”。

② ||当且仅当两个运算量的值都为“假”时，运算结果为“假”，否则为“真”。

③ ！当运算量的值为“真”时，运算结果为“假”；当运算量的值为“假”时，运算结果为“真”。

例如，假定 x=5，则(x>=0) && (x<10)的值为“真”，(x<-1)||(x>5)的值为“假”。

2. 逻辑运算符的运算优先级

(1)逻辑非的优先级最高，逻辑与次之，逻辑或最低，即：

！(非)→&&(与)→||(或)

(2)与其他种类运算符的优先关系

！→算术运算→关系运算→&&→||→赋值运算

表 4-1 为逻辑运算的“真值表”，表中列出了当 m 和 n 的值为不同组合时，各种逻辑运

算所得到的值。

表 4-1　逻辑运算真值表

m	n	! m	! n	m&&n	m‖n
非 0	非 0	0	0	1	1
非 0	0	0	1	0	1
0	非 0	1	0	0	1
0	0	1	1	0	0

3. 逻辑表达式

用逻辑运算符组成的式子称为逻辑表达式。逻辑表达式可以用来表示更为复杂的比较，例如数学式为：

10＜x≤20

可表示为 x>10&& x<=20。

与关系表达式一样，逻辑表达式的类型是逻辑型的，当满足时为“真”，取值为 1，否则为“假”，取值为 0。

编译程序在求解逻辑表达式的过程中，在处理含有 && 或‖的表达式，往往需要采用优化算法，在从左到右的运算中，只要结果一确定，就无需再继续往下运算。

① 对于‖运算，只要运算符的左边为 1，右边不需计算。例如表达式：

m++‖n++‖a++

如果 m、n、a 的初始值均为 1，在计算表达式时，由于都是或运算，只要出现了真，整个表达式的值就为真，故而计算了表达式 m++后就能确定整个表达式的值就为 1，因而 n++和 a++的运算就无需进行了。这个表达式求解后，m 的值为 2，而 n 和 a 的值仍为 1。

② 对于 && 运算，只要运算符的左边为 0，右边不需计算。例如表达式：

++m&&++n&&++a

若 m、n、a 的初始值均为－1，则++m 为 0，整个表达式的值为 0，n 和 a 自增不再运算，表达式执行完后，m 的值为 0，n 和 a 的值仍为－1。

在实际运用中，可以用逻辑表达式来表达复杂的条件。

例如，闰年的条件为：能被 4 整除但不能被 100 整除，或能被 400 整除。写出表示闰年的表达式。

解：用 year 表示年份，则闰年的表达式为：

(year%4==0&&year%100!=0)‖(year%400==0)

当该表达式值为“真”时，year 为闰年，否则为非闰年。

又例，如果要判断 ch 是否为字母，可以写成逻辑表达式为：

ch>='A'&&ch<='Z'‖ch>='a'&&ch<='z'

重点：在计算逻辑表达式时，如果在求得某个操作数的值后，可以判定整个逻辑表达式的真假，那么其后的操作数将不被计算。

4.2 if语句

if语句通常被称为选择语句、分支语句。它不再局限于顺序结构单一流程，而是用来对给定条件进行判定，根据其结果，从两种操作中选择其一。if语句用于实现分支结构。C语言提供了三种形式的if语句。

4.2.1 if语句的三种形式

1. 单分支结构

一般形式为：

if(表达式)语句；

例如：

```
if(x<0) printf("%d\n",x);
```

这种形式的if语句流程图，如图4-1所示。

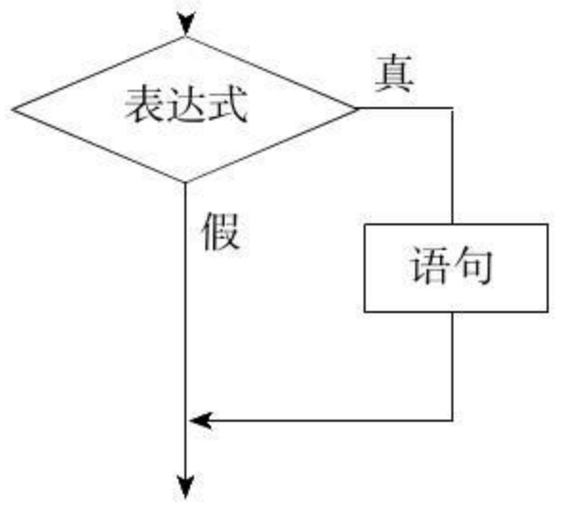


图4-1 单分支结构

在执行此种if语句时，首先对“表达式”进行判定，如果表达式的计算结果为“真”，则执行其后的“语句”，如果表达式的计算结果为“假”，则跳过其后的“语句”。

其中“表达式”通常是一个关系表达式，用于对两个值进行比较，如x>5、y<4等；或者是一个逻辑表达式，用于表示若干条件成立或不成立的关系，如m&&n等。

注意：“表达式”结果的真假取决于该表达式当前逻辑或关系运算的结果，而与先前运行的输入和状态无关，就是说，在程序的运行过程中，“表达式”随着表达式中变量值的变化，每次结果可能也不相同，可能为“真”，也可能为“假”。

图4-1中的“语句”可以是一条简单语句，也可以是一个复合语句。例如：

```
if (m<n)
   m=m+2;          /* 简单语句 */
if(m<n)
{ m=m+2;
   n=n+1;
}                  /* 复合语句 */
```

复合语句等效于一条语句，它常出现在函数体内。在一个函数体内可以并行出现多个复合语句，也可在一个复合语句中嵌套另一个复合语句。

2. 双分支结构

一般形式为：

if(表达式)语句1；

else 语句2；

if else语句的流程图如图4-2所示，其执行过程：

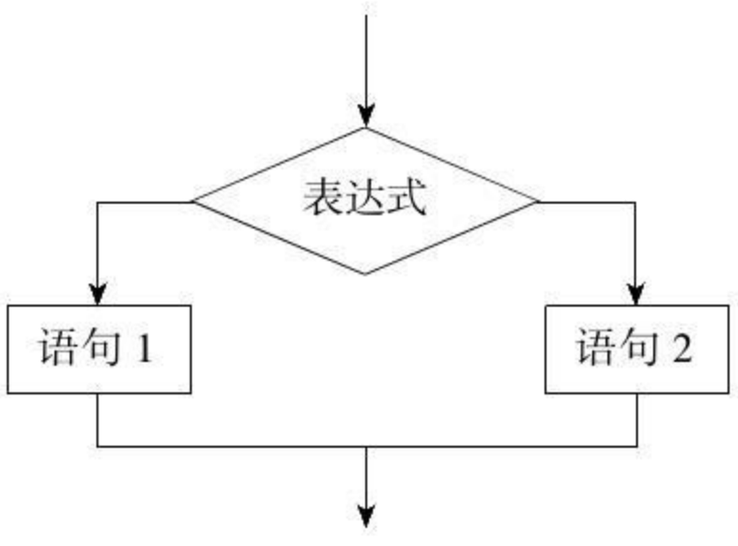


图4-2 双分支结构

先计算表达式值，当表达式为真，执行语句 1，否则执行语句 2。其中“语句”部分可以是简单语句或者复合语句。

例如：

```
if(m>=0)
  printf("m 是一个非负数");
else
  printf("m 是一个负数");
```

注意：在 if 后面的语句 1 和 else 后面的语句 2 部分，如果有多条语句，应用花括号“{ }”将这几条语句括起来，成为一条复合语句。

【例 4.1】　用 if 语句编写程序。

$$\begin{cases} r=a^2-b^2,s=\dfrac{a}{b} & a<b \\ r=b^2-a^2,s=\dfrac{a}{b}+4 & a\geqslant b \end{cases}$$

```
#include <stdio.h>
void main(void)
{ float a,b,r,s;
   scanf("%f,%f",&a,&b);
   if(a<b)
   { r=a*a-b*b;s=a/b;}
   else
   { r=b*b-a*a;s=a/b+4;}
   printf("r=%f,s=%f\n",r,s);
}
```

程序运行结果，如图 4-3 所示。

图 4-3　例 4.1 程序运行结果

3. 多分支结构

前面介绍的 if 语句允许用户选择是否执行某一操作，if else 语句允许用户在两个操作中选择其一执行。不过，在实际运用中可能面对两种以上的选择，因此把 if else 语句稍加扩展就能满足要求。其一般形式为：

```
if(表达式 1)   语句 1;
else if(表达式 2)   语句 2;
else if(表达式 3)   语句 3;
……
else if(表达式 n)   语句 n;
else      语句 n+1;
```

使用这种形式可以有几种不同的选择。

该形式的 if 语句功能是：自上而下判断条件，一旦发现条件为真，执行与之有关的语句，其余语句不被执行。如果没有条件为真，执行最后的 else 中的语句。

注意：最后一条 else 语句常作为缺省条件，即如果所有其他条件均测试失败，执行最后一条 else 语句，如果没有最后一条 else 语句，则不执行任何操作。

多分支结构的流程图如图 4-4 所示。

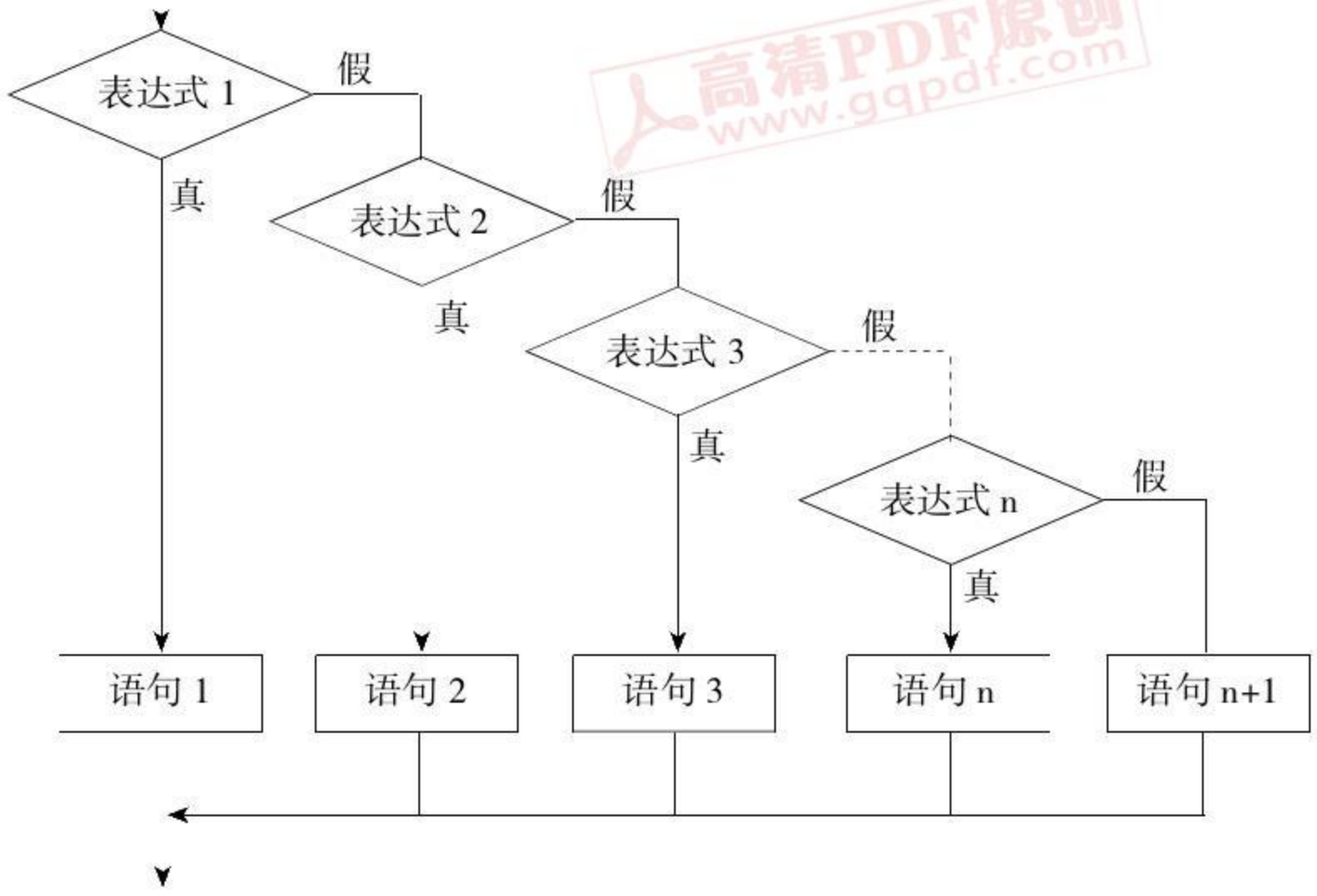


图 4-4 多分支结构

【例 4.2】 有一函数

$$m=\begin{cases} n & (n<1) \\ 2n-4 & (1\leqslant n<4) \\ 5n-9 & (n\geqslant 4) \end{cases}$$

写一程序，输入 n 值，计算并输出 m 的值。

```
#include <stdio.h>
void main(void)
{
   int,n;double m;
   printf("请输入 n 的值:");
   scanf("%d",&n);
   if(n<1)
     m=n;
   else if(n<4)
     m=2 * n-4;
   else
     m=5 * n-9;
   printf("m 值为:%.2 f\n",m);
}
```

程序运行结果，如图 4-5 所示。

图 4-5 例 4.2 程序运行结果

【解析】 程序在三个不同情况中选择一个，这取决于 n 的值。当 n 小于 1 时，选择 m＝n，当 n 介于 1 与 4 之间时，选择 m＝2n－4；当 n 大于等于 4 时，选择 m＝5n－9。因为该选择不再局限于两种选择，而是多种选择，故使用多分支 if－else－if 结构。

注意：用“%.2f”来控制 m 输出值的显示样式(保留小数点后两位)。

4.2.2　if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句，称为 if 语句的嵌套。一般形式如下：

```
if(表达式 A)
  if(表达式 B)语句 B1;
  else  语句 B2;
else
  if(表达式 C)语句 C1;
  else  语句 C2;
```

用流程图表示，如图 4－6 所示。

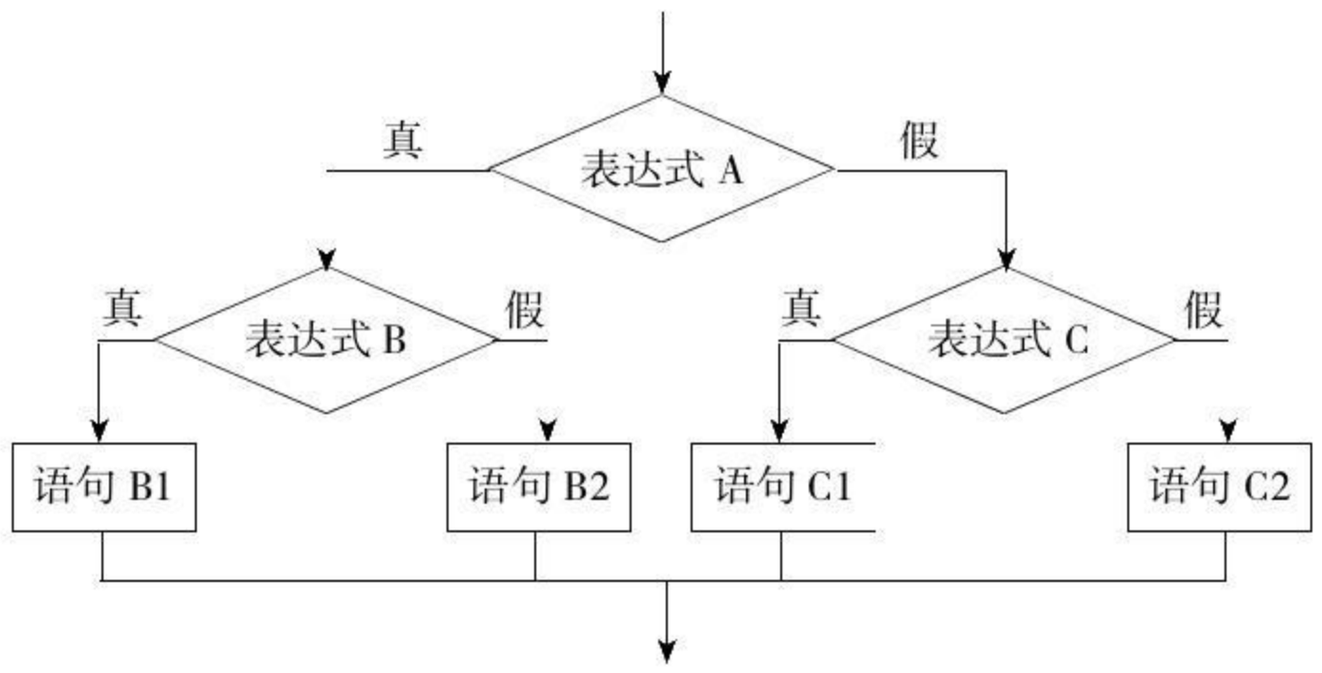


图 4－6　if 的嵌套结构

在 if 语句的嵌套结构中，一定注意 else 与 if 相匹配问题。C 语言的语法结构规定：else 总是与它前面最近的尚未配对的 if 匹配。例如：

```
if(a)
  if(b) printf("x");
  else  printf("y");
```

在此例中 else 与其前最近的 if(b)相对应。若要使 else 语句与 if(a)对应，就需使用花括号，如：

```
if(a)
{ if(b) printf("x");
}
else printf("y");
```

【例 4.3】 从键盘输入三个实数，输出其中最大者。

分析：要判断三者中最大者，首先要判断两者中的较大者，然后再与第三者比较。

```
#include <stdio.h>
void main(void)
{
   float x,y,z,imax;
   printf("请输入三个实数:");
   scanf("%f%f%f",&x,&y,&z);
   if(x>y)
     if(x>z)   imax=x;
     else   imax=z;
   else
     if(y>z)   imax=y;
     else   imax=z;
   printf("三者中最大值为:%f\n",imax);
}
```

图 4-7　例 4.3 程序运行结果

程序运行结果,如图 4-7 所示。

注意:else 与 if 的配对关系。嵌套 if 语句的书写格式,通常应采用缩排格式,使程序格式更清晰,但不会影响程序的编译。在 if 语句嵌套结构中,else 总是与其前面最近的 if 语句相匹配。

4.2.3　条件运算符

C 语言中有“?:”运算符,它是唯一的三目运算符,其功能与 if 语句极其相似,它由“?”和“:”组成,要求有三个运算对象。

表达式的一般形式为:

表达式 1 ? 表达式 2:表达式 3

其中,表达式 1 可以是整型、浮点型或字符型数,通常为关系表达式或逻辑表达式。表达式 2 和表达式 3 的类型可以不同,但条件表达式值的类型是两者中较高的类型。流程图如图 4-8 所示。

功能:先计算表达式 1,如果为真,则计算表达式 2 的值;如果为假,则计算表达式 3 的值。如:

m=(x>y)? x:y

表示将 x 和 y 中较大的值赋值给 m。当 x>y 结果为真时,将“:”前的 x 的值赋给变量 m,如果为假时,则将“:”后面的 y 的值赋给变量 m。

条件运算符的优先级如下:条件运算符的优先级高于赋值运算符,条件运算符的优先级低于关系运算符和算术运算符。

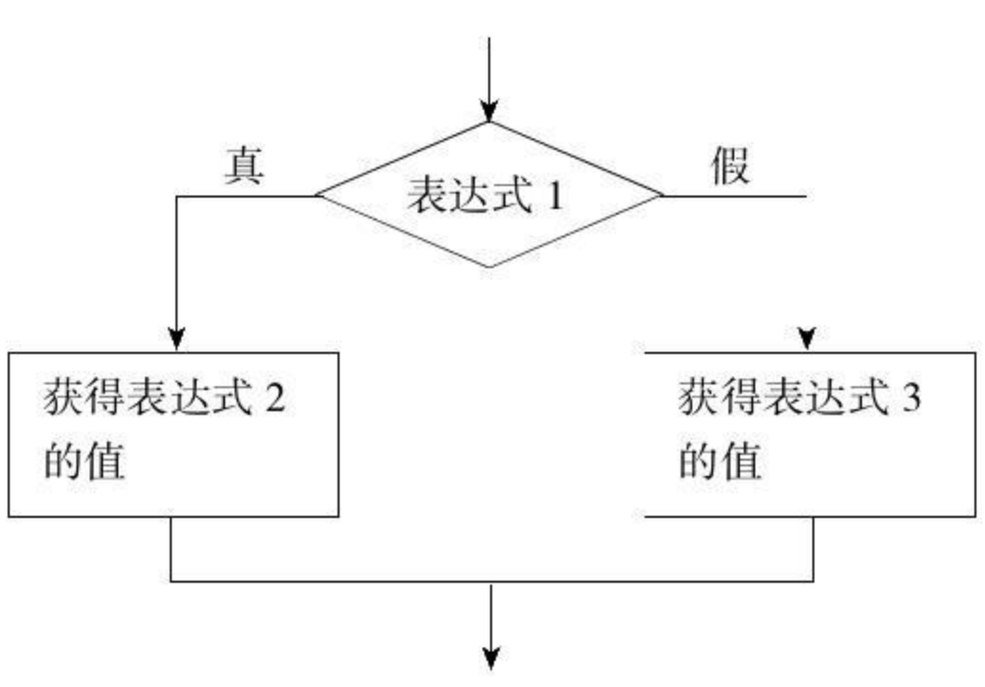


图 4-8　三项条件运算

条件运算符的结合性是自右向左。例如：

```
min=a<b? a:b          /*等价于 min=(a<b)? a:b*/
x>y? x:y-1            /*等价于(x>y)? x:(y-1)*/
a>b? a:b>d? c:d       /*等价于(a>b)? a:((b>d)? c:d)*/
```

注意：可以把函数调用放在条件运算符的表达式中(除非函数被说明为 void 类型)。

【例 4.4】 用三项条件运算符重写【例 4.3】。

```
#include <stdio.h>
void main(void)
{
   float x,y,z,imax;
   printf("请输入三个实数:");
   scanf("%f,%f,%f",&x,&y,&z);
   imax=x>y? x:y;
   imax=imax>z? imax:z;
   printf("三者中最大值为:%f\n",
imax);
}
```

图 4-9　例 4.4 程序运行结果

程序运行结果，如图 4-9 所示。

4.3　switch 语句

在实际应用中，程序常常面临多重选择，使用 if-else-if 语句可以实现多重情况。但是如果分支比较多，比如 5 个以上的分支时，使用过多的 if-else-if 语句就会使程序变得十分复杂，难以读懂，而且很容易产生错误。因而，当分支数目在三个以上时，可以使用 C 语言中提供的另一个多分支选择结构语句——switch 语句，也叫开关语句来实现。

switch 语句是用于多个分支选择的语句，可以根据不同的条件进行程序转移。它的一般形式如下：

```
switch(表达式)
{
   case 常量表达式 1:语句 1;break;
   case 常量表达式 2:语句 2;break;
   case 常量表达式 3:语句 3;break;
   ……
   case 常量表达式 n:语句 n;break;
   default:语句 n+1;
}
```

功能：

① 先计算 switch 后表达式值。

② 然后将该值与各表达式常量相比较，跳到与其值相等的 case 分支处，执行该 case 内的相关语句，一直执行到 break 语句；若是找遍所有的 case 标号，没有找到相等的情况，执行 default 后对应的语句。

③ 若这时没有 default 语句，则不执行任何分支(注：break 为跳转命令，详见第五章)。

说明：

(1)switch 语句与 if 语句不同，switch 测试表达式值与 case 后表达式常量是否相等，而 if 语句可进行表达式的真假判断。

(2)在同一个 switch 语句中，不允许两个 case 常量表达式有相同的值。

(3)switch 后面括号内的表达式可以是整型表达式或字符型表达式，也可以是枚举型数据。

(4)switch 结构允许多个 case 共同使用一个语句组。

例如：case'A':case'a':printf("优\n");break;

(5)从技术角度上说，break 语句在 switch 语句中是可选的，它是用来跳过后面的 case 语句，结束 switch 语句体，从而起到真正的分支作用。如果省略 break 语句，则继续执行下一个 case 语句直到遇到 break 或整个 switch 结构结束。

(6)“case 常量表达式”只是起语句标号作用，并不是在该处进行条件判断。在执行 switch 语句时，条件判断只是在开始进行一次，然后根据 switch 表达式的值找到匹配的入口标号，从此标号开始往下执行，此处不再进行判断了。

(7)default 标号后的语句只是在找不到匹配的值时才执行，即使它放在第一句，也是这样，不过一般我们习惯把它放在 switch 语句的最后一句。

可以使用嵌套的 switch 语句，并允许内外 case 标号相同。

【例 4.5】 根据考试成绩的等级(A、B、C、D)输出“优”、“良”等评语。

```
#include <stdio.h>
void main(void)
{
   char grade;
   printf("请输入成绩等级(A、B、C、D):");
   grade=getchar( );
   switch(grade)
   {
     case 'A': printf("优\n"); break;
     case 'B': printf("良\n"); break;
     case 'C': printf("及格\n"); break;
     case 'D': printf("不及格\n"); break;
     default:printf("输入错误\n");
   }
}
```


程序运行结果，如图 4-10 所示。

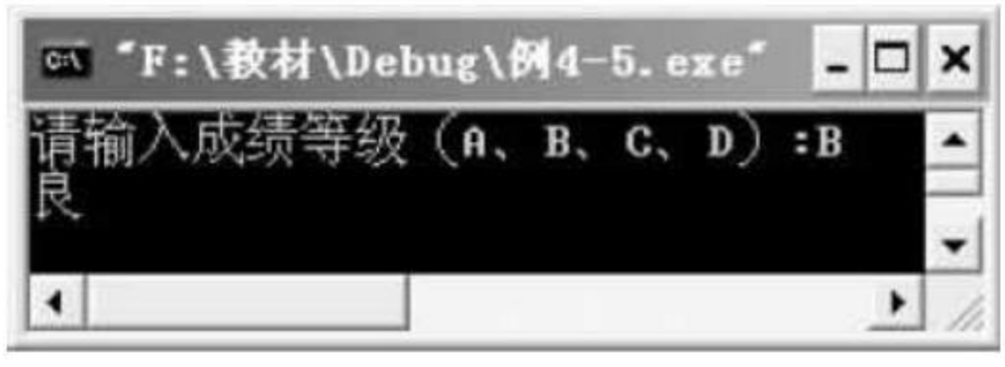


图 4-10 例 4.5 程序运行结果

【解析】 switch 语句的工作过程是：首

先对 switch 语句括号中的表达式进行计算(本例中是 grade 变量),然后程序再从上至下查找所有 case 中与表达式的值相匹配的常量表达式(本例中匹配的是 case 'C':),以此为入口,将程序转移到这个入口的第一条语句处,再顺序往下执行。若没有相匹配的常量表达式,这时如果 switch 语句中存在 default,那么程序就转到此处执行,如果没有 default 语句,程序就跳出 switch 结构,转而执行 switch 结构后面的语句。

switch 语句中,default 部分是可选的,不一定要设置,可视程序具体情况而定。一般而言,在多分支结构中总会出现“意外”情况,通常要设置 default 语句行,将不能与以上所有 case 常用表达式相匹配的情况,在 default 语句行中进行统一处理。程序中的 break 语句所起作用很重要,当程序执行 break 语句后,它就使程序流程跳出 switch 结构,并转而执行 switch 结构后面的第一条语句。如果程序中没有 break 语句,将会继续执行下一个 case 语句后面的程序段,一直到出现 break 语句或到 switch 结构结束为止。

注意:一般情况下,根据需要,每个 case 后添加一个 break 语句。否则执行的结果将不满足要求。

4.4　例题精解

【例 4.6】 在以下各组运算符中,优先级最高的运算符分别为

(1)　A)?:　　B)++　　C)&&　　D)+=

(2)　A)*=　　B)>=　　C)(类型)　　D),

(3)　A)||　　B)%　　C)!　　D)==

(4)　A)=　　B)!=　　C)*(乘号)　　D)( )

【解析】 通过本题读者应能弄清 C 语言中常用的几类运算符的优先级。C 语言中各类运算符的优先级按由高到低的次序可大致归纳如下:

初等运算符(包括( ),[ ]等)→单目运算符(包括!,++,(类型))等,其中(类型)代表强制类型转换运算符)→算术运算符(先乘除,后加减)→关系运算符→逻辑运算符(不包括!)→条件运算符→赋值运算符(包括=,+=,*=等)→逗号运算符。

由此可知,题中各组运算符的优先级顺序(由高至低)如下所示:

第(1)组:　++→ &&→ ?:→ +=

第(2)组:　(类型)→ >=→ *=→,

第(3)组:　! → %→==→||

第(4)组:　( )→ *→!=→=

因此,正确答案是:(1)B;　(2)C;　(3)C;　(4)D。

【例 4.7】 商场迎春促销,以打折吸引顾客,其打折情况如下:

购买量<20　　原价销售

购买量<50　　打 9 折销售

购买量<100　　打 8 折销售

购买量>=100　　打 7 折销售

编程实现:输入商品购买量和单价,求应缴金额。

```
#include <stdio.h>
void main(void)
{
   int n;        /* 商品的购买量 */
   float price;double money,m;        /* 商品的单价和应付金额及商品打折率 */
   printf("请输入商品购买量和单价:");
   scanf("%d%f",&n,&price);
   if(n<20)
     m=1.0;
   else if(n<50)
     m=0.9;
   else if(n<100)
     m=0.8;
   else
     m=0.7;
   money=price * n * m;
   printf("应付%.2f 元\n",money);
}
```

程序运行结果,如图 4-11 所示。

【解析】 据题意可知,由购买量决定打折情况,变量 m 代表打折率,变量 n 代表购买量。①当 n 小于 20 时,m=1.0。②当 n 介于 20 与 50 之间时,m=0.9。③当 n 介于 50 与 100 之间时,m=0.8。④当 n 大于或等于 100 时,m=0.7。有多个选择,故可用 if-else-if 结构。

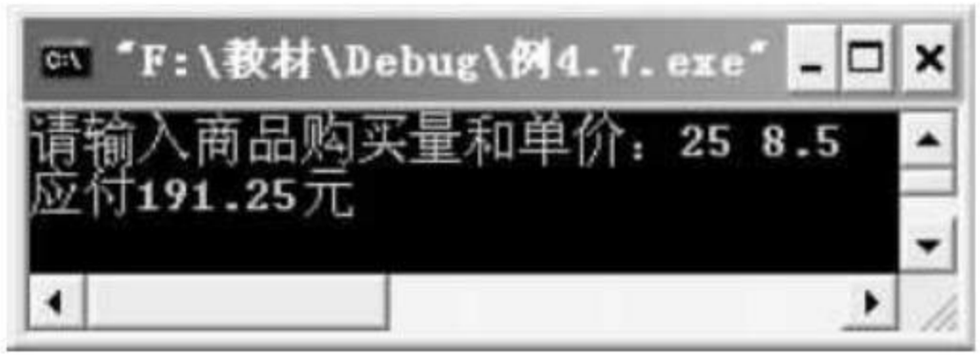


图 4-11 例 4.7 程序运行结果

【例 4.8】 编写程序判断某一年是否为闰年。

```
#include <stdio.h>
void main(void)
{
   int year;
   printf("请输入年份:");
   scanf("%d",&year);
   if(((year%4==0)&&(year%100!=0))||(year%400==0))
     printf("%d 是闰年.\n",year);
   else
     printf("%d 不是闰年.\n",year);
}
```

程序运行结果,如图 4-12 所示。

图 4-12　例 4.8 程序运行结果

【解析】 闰年的条件符合下列两个条件之一：能被 4 整除，但不能被 100 整除；能被 400 整除。

写成逻辑表达式为：

(year%4==0)&&(year%100!=0)

(year%400==0)

两者合并为一个式子：(year%4==0)&&(year%100!=0)||(year%400==0)

【例 4.9】 输入年份，输出该年的属相。

```
#include <stdio.h>
void main(void)
{
   int year,m;
   printf("请输入年份:\n");
   scanf("%d",&year);
   m=2012-year;
   m%=12;
   switch(m)
   {
      case 0:printf("%d 年属鼠\n",year);break;
      case 1:printf("%d 年属牛\n",year);break;
      case 2:printf("%d 年属虎\n",year);break;
      case 3:printf("%d 年属兔\n",year);break;
      case 4:printf("%d 年属龙\n",year);break;
      case 5:printf("%d 年属蛇\n",year);break;
      case 6:printf("%d 年属马\n",year);break;
      case 7:printf("%d 年属羊\n",year);break;
      case 8:printf("%d 年属猴\n",year);break;
      case 9:printf("%d 年属鸡\n",year);break;
      case 10:printf("%d 年属狗\n",year);break;
      case 11:printf("%d 年属猪\n",year);
   }
}
```

程序运行结果，如图 4-13 所示。

图 4-13　例 4.9 程序运行结果

【解析】 十二生肖的顺序为：鼠、牛、虎、兔、龙、蛇、马、羊、猴、鸡、狗、猪，只要知道某年的出生年份，就可以推出其生肖。如今年是2012年，减去出生年份，再计算除以12后的余数，如果余0，就是鼠年，余1就是牛年…依次类推。

4.5 本章小结

本章是C语言根据某种条件的成立与否，而采用不同的程序段进行处理的程序结构称为选择结构。选择结构是C语言中一种重要的语句结构，它体现了程序的逻辑判断能力。C语言提供了多种形式的条件语句以构成选择(分支)结构。

(1)if语句主要用于单分支结构；

(2)if－else语句主要用于双分支结构；

(3)if－else－if语和switch语句用于多分支结构。

选择结构离不开逻辑判断，关系运算和逻辑运算正体现了这种逻辑判断能力。选择结构的控制条件通常用关系表达式或逻辑表达式构造，也可以用一般表达式表示。因为表达式的值非0即为“真”，0即为“假”。语句的选择和使用对整个程序的流畅、简洁、易懂起到关键的作用，因而要掌握每个语句格式的格式和功能。

习　题

一、选择题

1. 下列运算符中优先级最高的是_____。

A）！　　B）％　　C）－＝　　D）&

2. 若要求在if后一对圆括号中的表达式，表示a不等于0时的值为“真”，则能正确表示这一关系的表达式是_____。

A）a！＝0　　B）！a　　C）a＝0　　D）a

3. 在C语言中，能代表逻辑值“真”的是_____。

A）true　　B）大于0的数　　C）非0整数　　D）非0的数

4. 若给定条件表达式(M)？(a＋＋)：(q－－)，则其中表达式M与_____等价。

A）M＝＝0　　B）M＝＝1

C）M!＝0　　D）M!＝1

5. 已知变量a代表整数，则不能正确表达数学关系：10＜a＜15的表达式是_____。

A）10＜a＜15

B）a＝＝11‖a＝＝12‖a＝＝13‖a＝＝14

C）a＞10&&a＜15

D）！(a＜＝10)&&！(a＞＝15)

6. 请选出能表示：x的值在－10到－1内，或在10到1范围内时，值为“真”的表达式_____。

A）(x＞＝－10)&&(x＜＝－1)‖(x＜＝10)&&(x＞＝1)

B）！(x＞＝－10)&(x＜＝－1)‖(x＜＝10)&&(x＞＝1)

C) x>-10||x<-1 &&x>10||x<1

D) (x<-10)||(x>-1 && x<1)||(x>10)

7. 设 a、b 和 c 都是 int 型变量，且 a=3，b=4，c=5；则以下的表达式中值为 0 表达式是_____。

A) a&&b　　B) a||b+c&&b-c

C) a<=b　　D) !((a<b)&& ! c||1)

8. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{
  int a=5,b=4,c=6,d;
  printf("%d\n",d=a>b? (a<c? a : c) : b);
}
```

A) 5　　B) 4　　C) 6　　D) 不确定

9. 对程序段：

```
if(m>n) x=1;
else if(m>0)x=2;
else x=3;
```

要得到 x=2 的结果，满足条件的一组数是_____。

A) m=-1,n=-2　　B) m=-2,n=-1

C) m=2;n=1　　D) m=2,n=3

10. 若变量都已正确说明，则以下程序段_____。

```
a=2;
b=3;
printf(a>b?"***a=%d":"###b=%d",a,b);
```

A) 没有正确的输出格式控制　　B) 输出为：***a=2

C) 输出为：###b=2　　D) 输出为：***a=2###b=2

11. 执行下列程序段后，x、y、z 的值分别为_____。

```
int x=10,y=20,z=30;
if(x>y)z=x;x=y;y=z;
```

A) 10,20,30　　B) 20,30,30　　C) 20,30,10　　D) 20,30,20

12. 设 x,y,z 是 int 型变量，且 x=3,y=4,z=5，则下面表达式中值为 0 的是_____。

A) 'x'&&'y'　　B) x<=y

C) x||y+z&&y-z　　D) !(x<y)

13. 判断 char 型变量 c1 是否为小写字母的正确表达式为_____。

A) 'a'<=c1<='z'　　B) (c1>=a)&&(c1<=z)

C) ('a'>=c1)||('z'<=c1)　　D) (c1>='a')&&(c1<='z')

14. 已知 x=43,ch='A',y=0；则表达式(x>=y&&ch<'B'&& ! y)的值是_____。

A) 0　　B) 语法错　　C) 1　　D) '假'

二、填空题

1. C语言的关系运算符中属于高优先级组的有：____、____、____、____，属于低优先级组的有____、____。

2. A&&B 的含义为 _____；A‖B 的含义为 _____；!A 的含义为 _____。三种逻辑运算符优先级为 _____。

3. 若 a=7，b=8，c=9，则表达式 a+b<c&&a>b 的值为 _____。

4. 设 a=2，b=3，c=4，逻辑表达式!(a+b)+c-1&&b+c/2 的值 _____；a+b>c&&b==c 的值为 _____；!(a+b)&&c‖1 的值为 _____。

5. 以下程序实现从键盘输入两数 a 和 b，输出其中大者(不考虑相等的情况)，请完善。

```
#include <stdio.h>
void main(void)
{     (1)     ;                                    (1)_____
  scanf ("%d%d",&a,&b);
  if(   (2)   )                                    (2)_____
    printf("最大值为%d",a);
  else
    printf("最大值为%d",b);
}
```

6. 以下程序判定某一年为闰年，请完善。

```
#include <stdio.h>
void main(void)
{ int year;
  printf("请输入年数:");
  scanf("%d",&year);
  if((year%4==0)&&   (1)   )                       (1)_____
    printf("%d 年是闰年",year);
  else if (     (2)     )                          (2)_____
    printf("%d 年是闰年",year);
  else
    printf("%d 年是平年",year);
}
```

三、应用题

1. 分析程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{ int   i=20;
  switch (i)
  {
    case 19: i+=1;
```

```
    case 20: i+=1;
    case 21: i+=1;
    case 22: i+=1;
  }
  printf("i=%d",i);
}
```

2. 分析程序,写出运行结果。

```
#include <stdio.h>
void main(void)
{ int x=1,y=0;
  switch(x)
  { case 1:
    switch(y)
    { case 0:printf("**1**\n");break;
      case 1:printf("**2**\n");break;
    }
    case 2: printf("**3**\n");
  }
}
```

3. 分析程序,写出运行结果。

```
#include <stdio.h>
void main(void)
{ int x=1,y=0,a=0,b=0;
  switch(x)
  { case 1:switch(y)
           { case 0:a++;break;
             case 1:b++;break;
           }
    case 2:a++;b++;
  }
  printf("a=%d,b=%d\n",a,b);
}
```

4. 分析程序,写出运行结果。

```
#include <stdio.h>
void main(void)
{ int w=4,x=3,y=2,z=1;
  if (x>y&&(z==w))
    printf("%d\n",(w<x? w:z<y? z:x));
  else
```

```
    printf("%d\n",(w>x? w:z>y? z:x));
}
```

四、编写程序题

1. 已知

$$y=\begin{cases}1 & x>0\\0 & x=0\\-1 & x<0\end{cases}$$

编一程序，输入一个 x 值，输出 y 值。

2. 假设在运输货物时，每吨公里运费 p 与距离 s 有关，路途越远，每吨公里运价越低，其公式如下：

$$p=\begin{cases}8 & (s<100)\\6 & (100\leqslant s<150)\\4 & (150\leqslant s<200)\\2 & (200\leqslant s<250)\\1 & (250\geqslant s)\end{cases}$$

如所付的运费超过 250 元，再给予 0.95 折优惠。从键盘输入货物吨数，运输公里数，求应付的运费。

3. 从键盘上输入 a，b，c 三个整数，输出其中最小者。

4. 编写程序，判断从键盘输入的一个字符，并按下列要求输出：

(1)若该字符是数字，输出字符串"0～9"；

(2)若该字符是大写字母，输出字符串"A～Z"；

(3)若该字符是小写字母，输出字符串"a～z"；

(4)若该字符是其他字符，输出字符串"!，@，……"。

第 5 章　循环结构程序设计

【教学提示】

循环结构是结构化程序设计的基本结构之一。本章主要介绍在 C 语言中实现循环的三种语句：while 语句、do-while 语句和 for 语句及其每种语句的格式、执行过程和实际应用，同时也介绍了循环的嵌套、break 语句和 continue 语句。熟练掌握循环结构的编程方法是程序设计的基本要求。

【核心概念】

循环结构　循环条件　循环体　当型循环　直到型循环　循环嵌套

5.1　概　述

循环结构是结构化程序设计的基本结构之一，它与顺序结构、选择结构共同作为各种复杂程序的基本构造单元。所谓循环结构就是当给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。利用循环结构处理各类重复操作既简单又方便。因此熟练掌握循环结构语句的格式和功能，并能根据循环结构的要求正确选取循环语句来实现循环是程序设计最基本的要求。

在 C 语言中，通常用 while 语句、do-while 语句及 for 语句三种循环语句来实现循环结构。

5.2　while 语句

5.2.1　while 循环的一般形式

while 语句是一种先判断、后执行的循环语句，用来实现“当型”循环结构。其一般形式为：

while（表达式）循环体；

当表达式的值为“真”时，执行循环体语句，为“假”时退出循环，执行循环的后继语句。例如：

```
int i=1,sum=0;
while(i<=100)
{ sum=sum+i;
  i++;
}
```

该程序段用来实现 1 到 100 的累加和，结果放在 sum 中。

几点说明：

(1)while 是 c 语言的关键字；

(2)圆括号中的表达式，可以是 c 语言中任意合法的表达式，其值是一个逻辑值，用于控

制循环体是否被执行；

(3)while 语句后没有分号(;)；

(4)循环体可以包含一个语句，也可以有多个语句组成，若有多个语句，应用一对大括号括起来，组成复合语句；

(5)循环体中应有使表达式趋假语句，从而使循环可以结束，否则会产生“死循环”。为了保证循环体可以结束，通常采用每循环一次向某方向改变表达式的值，使得表达式的值最终可以为假。还有一种方法是在某种条件下，强行从循环体中跳出(如使用 break 语句，该语句在 5.6 节介绍)。

5.2.2 while 循环执行过程

while 语句是根据表达式的值决定程序的流向，当为非 0(“真”)时，执行循环体，否则退出循环，执行其后的语句。其流程图如图 5-1 所示。

具体执行过程如下：

(1)计算 while 后圆括号中表达式的值。当值为非零时，执行(2)；当值为零时执行(4)；

(2)执行循环体内的语句；

(3)转向执行(1)；

(4)结束 while 循环，去执行 while 循环后的语句。

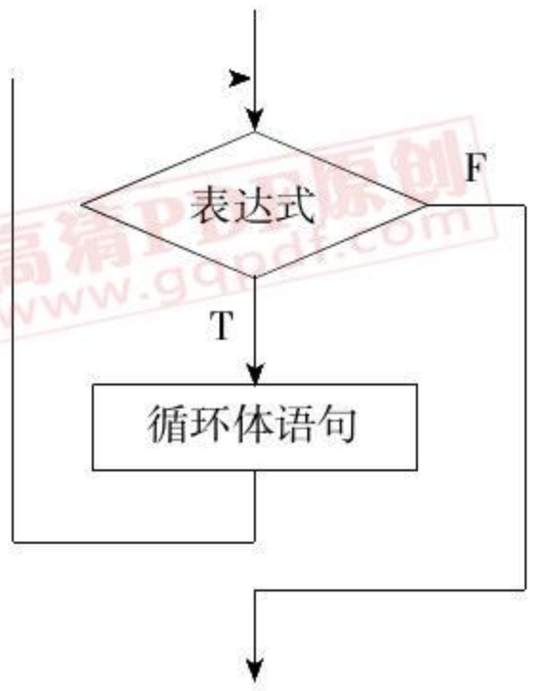


图 5-1 while 循环的执行过程

while 循环的特点是：先判断表达式，后执行语句。表达式可以是赋值表达式、关系表达式和逻辑表达式。表达式用来确定是否继续循环还是结束循环。如果一开始表达式值为 0，将一次也不执行循环体。

【例 5.1】 编写程序，求 1+2+3+…100 的值。

这是一个求多个有规律的数相加的问题，为此主要解决两个问题，第一是获取数，第二是求和。

获取数：第一个数为 1，其后的数都是前一个数加 1，直至 100。因此可以在循环体中使用一个变量 i，每循环一次使 i 增 1，即用 i++实现，直到 i 的值超过 100，这样可以获取所要的每个数。

求累加和：可用一个变量 sum 来存放和，给 sum 赋初值 0，第一次执行循环体时，sum+1 赋给 sum，i 的值变为 2，i<=100 为真，继续执行循环体，sum+2 赋给 sum，i 变为 3，i<=100 成立，继续执行循环体，依次类推。直至循环结束，最后 sum 中存放所求的累加和。

```
#include <stdio.h>
void main(void)
{ int i,sum;
  i=1;
  sum=0;                    /*给 sum 赋初值 0*/
  while(i<=100)             /*当 i 小于等于 100 时执行循环体*/
  { sum=sum+i;              /*累加 i 值*/
    i++;                    /*i 增加 1,使表达式趋假语句*/
```

```
    }
    printf("sum＝%d\n",sum);
}
```

程序運行結果,如圖 5－2 所示。

圖 5－2　例 5.1 程序運行結果

在源程序的基礎上,若把循環體改爲:

```
{ i＋＋;
  sum＝sum＋i;
}
```

請同學們分析一下結果爲多少? 如果要改變成原來的結果,程序如何修改?

【例 5.2】　求 $\sum_{n=1}^{10} n!$

求 1 到 10 的階乘的累加和。即求 1!＋2!＋…＋10! 。

```
#include <stdio.h>
void main(void)
{ int n＝0;
  long t＝1,sum＝0;
  while(n<10)
  { n＋＋;
    t *＝n;                    /*t 中存放的是 n! */
    sum＋＝t;
  }
  printf("%ld\n",sum);
}
```


圖 5－3　例 5.2 程序運行結果

程序運行結果,如圖 5－3 所示。

該程序中 n 的初值爲 0,每執行一次循環體,n 加 1,變量 t 中存放的是該 n 的階乘,sum 中存的是階乘的累加和,n 的變化是從 1 到 10,從而滿足題意。

5.3　do while 语句

5.3.1　do while 语句一般形式

do while 語句也是一種循環語句,是實現"直到型"循環,其一般形式爲:

```
do
{
   循環體
} while (表達式);
```

該語句是先執行一次循環體語句,然后判別表達式,直到條件爲假才退出循環。例如:

```
i＝0;
sum＝0;
```

```
do
{ i++;
  sum+=i;
}while(i<100);
```

实现1到100的累加,结果放在sum中。

说明:

(1)do是c语言的关键字,必须和while联合使用,do后不可有“;”,在while(表达式)后的“;”不可丢,它表示do while语句的结束;

(2)圆括号中的表达式,可以是c语言中任意合法的表达式,其值是一个逻辑值,用于控制循环体是否执行;

(3)循环体可以包含一个语句,也可以有多个语句组成,若有多个语句应用大括号括起来,组成复合语句;

(4)循环体中应有改变表达式值的语句从而使循环可以结束,否则会产生“死循环”。为了保证循环体可以结束,通常采用每循环一次向某方向改变表达式的值,使得表达式的值最终可以为假。还有一种方法是在某种条件下,强行从循环体中跳出(如果使用break语句,该语句在5.6中介绍)。

5.3.2 do while循环的执行过程

do while语句是先执行一次指定的循环体语句,然后判别表达式,当表达式的值为非0(“真”)时,返回重新执行循环体。如此反复,直到表达式的值为0(“假”)时为止,此时循环结束,其流程图如图5-4所示。

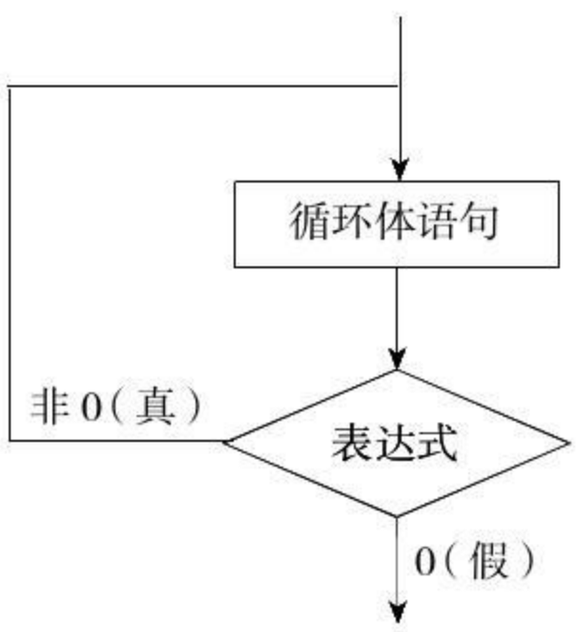


图5-4 do while循环的执行过程

具体执行过程如下:

(1)执行do和while之间的循环体语句;

(2)计算while后表达式的值,当值为非0时转去执行(1);若值为0时执行(3);

(3)结束循环,去执行do while循环后的语句。

do while语句的特点:先执行循环体,然后判断循环条件是否成立。

【例5.3】 用do while语句求1+2+3+…100的值。

```
void main(void)
{ int i=1,sum=0;
  do
  {  sum+=i;
     i++;
  }while(i<=100);
  printf("sum=%d\n",sum);
}
```

程序运行结果,如图5-5所示。

图5-5 例5.3程序运行结果

通过例5.1和例5.2可以看出，由do while构成的循环与while循环十分相似，同一问题可以分别用while语句和do while语句处理，两者可以相互转换，但它们之间有重要的区别：while循环是先判断，后执行；而do while是先执行，后判断。

5.4　for语句

5.4.1　for语句的一般形式

for语句也是一种实现循环的语句，且使用最为灵活，其功能是把某些语句重复执行若干次，不仅可以用于循环次数确定的情况，而且可以用于循环次数不确定的情况，因此，它完全可以代替前面所讲述的while语句。它的一般形式为：

for（表达式1；表达式2；表达式3）

**　　循环体**

“表达式1”通常用来给循环变量赋初值。当然，也允许在for语句外给循环变量赋初值，此时可以省略该表达式。“表达式2”通常是循环条件，以便决定是否继续执行循环体。一般为关系表达式或逻辑表达式。“表达式3”通常可用来修改循环变量的值，一般是赋值语句。

例如：for（k＝0；k＜10；k＋＋）

　　printf（"＊"）；

实现在一行上输出10个“＊”号。

5.4.2　for循环的执行过程

该语句的执行过程比较复杂，其流程图如图5－6所示。

具体执行过程如下：

（1）先求表达式1（相当于初始化，仅执行一次）；

（2）求解表达式2，若值为非0（“真”），执行循环体后执行（3）；若值为0，终止循环，转到（4）；

（3）求解表达式3；转回（2），进行下一次循环操作；

（4）退出循环，执行for语句后面的语句。

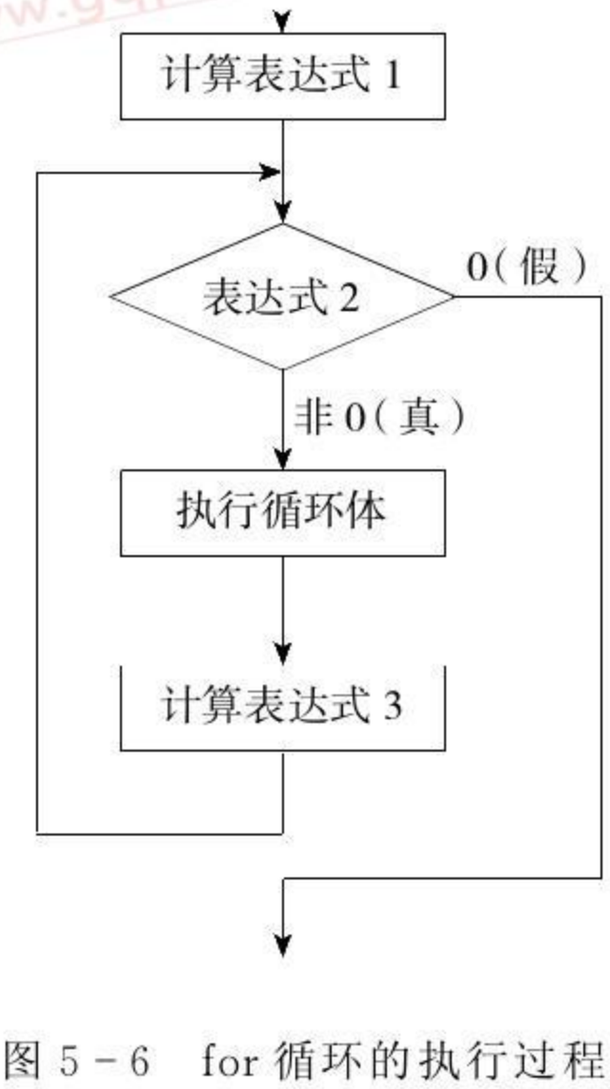


图5－6　for循环的执行过程

【例5.4】　用for语句求1＋2＋3＋…100的值。

```
#include <stdio.h>
void main(void)
{
   int i,sum=0;
   for(i=1;i<=100;i++)
     sum+=i;
   printf("sum=%d\n",sum);
}
```

程序运行结果，如图5－7所示。

图5－7　例5.4程序运行结果

与 while 比较 for 语句中表达式 1 提供了初始化，而 while 语句则必须在外面指定初值。可以把 for 语句的执行改为：

```
表达式 1;
while(表达式 2)
{ 语句
  表达式 3;
}
```

上面【例 5.4】求和可写为：

```
#include <stdio.h>
void main(void)
{
  int i,sum=0;
  i=1;                    /*表达式 1*/
  while(i<=100)           /*表达式 2*/
  { sum+=i;
    i++;                  /*表达式 3*/
  }
  printf("sum=%d\n",sum);
}
```

5.4.3 有关 for 语句的说明

1. 省略表达式 1

因表达式 1 是置初值，整个循环只做一次，可将其放在循环之前，省略表达式 1，但必须保留其后的分号。其一般形式为：

for(;表达式 2;表达式 3) 循环体

例如，将【例 5.4】可改为：

```
#include <stdio.h>
void main(void)
{
  int i=1,sum=0;
  for(;i<=100;i++)
    sum+=i;
  printf("sum=%d\n",sum);
}
```


2. 省略表达式 2

表达式 2 是判断是否执行循环的表达式。如果省略(其后分号不能省)，即不判断循环条件，循环无终止执行，也就是认为表达式 2 始终为“真”。在这种情况下，如果循环体中没有能终止循环的语句，将成为死循环，为了使程序仍能正确终止，一般循环体中应有判别语句，即用 break 语句退出循环。其一般形式为：

for(表达式 1;;表达式 3) 循环体

例如,将【例 5.4】可改为:

```
#include <stdio.h>
void main(void)
{
   int i,sum=0;
   for(i=1;;i++)
   {
     if (i>100) break;        /* 当 i>100 时用 break 退出循环 */
     sum+=i;
   }
   printf("sum=%d\n",sum);
}
```

3. 省略表达式 3

因表达式 3 在执行循环体之后再执行,所以可将其放入循环体中,从而省略表达式 3,效果一样,但其前的分号不能省。其一般形式为:

for(表达式 1;表达式 2;) 循环体

例如,将【例 5.4】可改为:

```
#include <stdio.h>
void main(void)
{
   int i,sum=0;
   for(i=1;i<=100;)
     sum+=i++;
   printf("sum=%d\n",sum);
}
```

4. 同时省略若干表达式

最极端情况,省略全部表达式,将其应有功能由其他语句来代替。其一般形式为:

for(;;) 循环体

例如,将【例 5.4】可改为:

```
#include <stdio.h>
void main(void)
{
   int i=1,sum=0;
   for(;;)
   {
     if (i>100) break;
     sum+=i++;
   }
```

```
    printf("sum=%d\n",sum);
}
```

5. 省略循环体

当循环体比较简单时,可把其放到表达式 3 中去,从而使循环体本身成为空语句。【例 5.4】可改为:

```
#include <stdio.h>
void main(void)
{
   int i,sum=0;
   for(i=1;i<=100;sum+=i++);
   printf("sum=%d\n",sum);
}
```

注意:空语句是一个分号,跟在 for 的括号之后形成一个完整的 for 语句。

6. for 后一对括号中的表达式可以是任意有效的 C 语言表达式。如:

```
for(sum=0,i=1;i<=100;sum=sum+i,i++)
```

表达式 1 和表达式 3 都是一个逗号表达式。

【例 5.5】 求 n!,即计算 1×2×3×…×n 的值。

```
#include <stdio.h>
void main(void)
{ int  i,n;
   long s=1;
   printf("Enter  n:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
     s=s*i;
   printf("%d!=%ld\n",n,s);
}
```

图 5-8 例 5.5 程序运行结果

程序运行结果,如图 5-8 所示。

【解析】 求阶乘的算法与累加一样,程序中 i 从 1 变化到 n,每次增 1 用 i++实现,s 的初值只能赋 1 不能为 0,否则结果为 0。当 i 为 1 时,进行 1*1 运算,结果 1 赋给 s,当 i 为 2 时,进行 1*2 运算,结果 2 赋给 s,当 i 为 3 时,进行 2*3 运算,结果赋给 s,依次类推,最终将 1*2*3*4*……n 值存入 s 中。

重点:♦while 语句是先判断条件,后执行循环体,所以循环体有可能一次也不执行。

♦do-while 语句是先执行一次循环体,再判断条件,所以循环体,至少被执行一次。

♦注意 for 语句中表达式 1、表达式 2、表达式 3 及循环体的执行顺序。

5.5　循环嵌套

在一个循环体内又完整地包含另一个循环，称为循环嵌套。内嵌的循环中还可以嵌套循环，这就是多层嵌套（又称多重循环）。前面介绍的三种类型循环（while 循环、do while 循环和 for 循环）可以互相嵌套。例如：

```
while()
{ ……
   for()
   {
     while() {……}
   }
}
```

是三重循环，而

```
for()
{ do
   {……
     for() {…}
   }while();
}
```

也是三重循环。

注意：嵌套只能完全包含，不能出现交叉现象。

【例 5.6】　使用双层 for 循环打印下面的图形。

```
    * * * * *
     * * * * *
      * * * * *
       * * * * *
```

```
#include <stdio.h>
void main(void)
{ int   k,i,j;
   for(i=0;i<=3;i++)                          /*i 是外循环变量用于控制行数*/
   {
     for(k=1;k<=i;k++) printf(" ");           /*k 是内循环变量用于控制每行的空格数*/
     for(j=0;j<=4;j++) printf("*");           /*j 是内循环变量用于控制每行中的星号*个数*/
     printf("\n");
   }
}
```

程序运行结果，如图 5-9 所示。

图 5-9 例 5.6 程序运行结果

上述程序中由 i 控制的 for 循环中内嵌两个平行的 for 循环，由变量 k 控制的循环实现每行开始的空格数，由于每行的空格数不同，第一行无空格，第二行 1 个空格，第三行 2 个空格依次类推。所以循环的结束条件和外循环的控制变量 i 有关即 k<=i。由变量 j 控制的循环实现连续输出 5 个“*”号。

双重循环中 i、k 和 j 值的变化规律，见表 5-1 所列。

表 5-1 i、k 和 j 值的变化规律

i 的变化	k 的变化	j 的变化(当等于 5 时退出循环)
0	1	0,1,2,3,4,5
1	1,2	0,1,2,3,4,5
2	1,2,3	0,1,2,3,4,5
3	当 i 等于 3 时退出循环	

【例 5.7】 “百鸡问题”是我国古代数学家张丘建在他编写的《算经》里提出的一个不定方程问题，即“鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一，百钱买百鸡，问鸡翁、母、雏各几何？”

```
#include <stdio.h>
void main(void)
{
   int x,y,z;
   for(x=0;x<=20;x++)
   {
     for(y=0;y<=34;y++)
     {
       z=100-x-y;
       if ((z%3==0)&&((5*x+3*y+z/3)==100))
          printf("公鸡=%d\t 母鸡=%d\t  小鸡=%d\n",x,y,z);
     }
   }
}
```

程序运行结果，如图 5-10 所示。

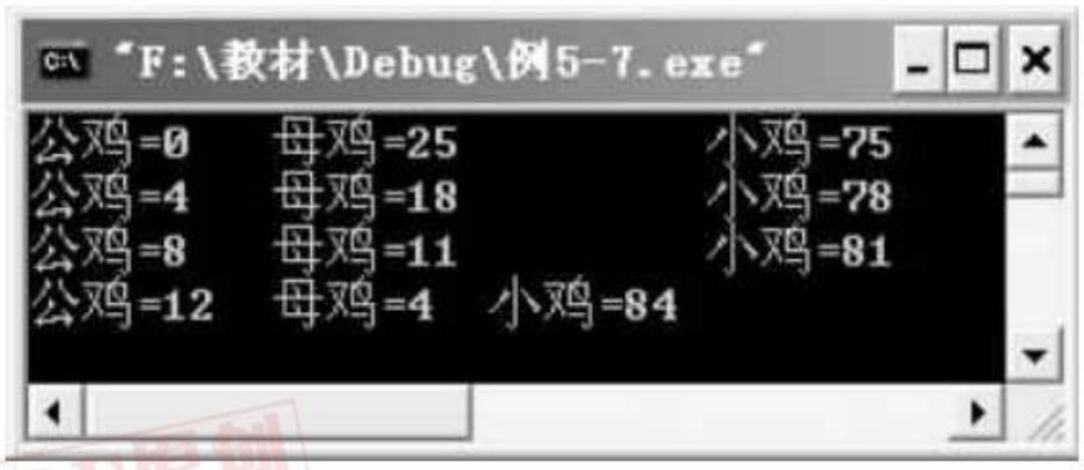


图 5-10　例 5.7 程序运行结果

【解析】 题中的含义为：一只公鸡价值 5 钱，一只母鸡价值 3 钱，三只小鸡价值 1 钱，现用 100 钱，去购买 100 只鸡的方法。在此假设公鸡、母鸡和小鸡各为 x、y 和 z 只，若全买公鸡，最多买 20 只，因此 x 的变化范围在 0～20 之间。同时，y 的变化范围在 0～34 之间，可得到的不定方程为：5x＋3y＋z/3＝100。

这是一个穷举法，外循环变量 x 在从 0～20 变化时，内循环变量 y 在 0～34 之间变化，去找满足不定方程的 z 值，同时保证 z 是 3 的倍数。如：当 x＝0，y 分别在 0 到 34 之间取值，再通过方程(5 * x＋3 * y＋z/3＝＝100)&&(z%3＝＝0)是否成立来确定，若成立就是要求的一组解；然后，当 x＝1，再去判断，依次类推。

5.6　break 语句、continue 语句和 goto 语句与标号

5.6.1　break 语句

break 语句只用在 switch 语句或循环语句中，其作用是无条件跳出 switch 语句或跳出本层循环，转去执行后面的程序。break 语句不需要语句标号与之配合。

break 语句的一般形式为：

break;

使用 break 语句可以使循环语句有多个出口，使程序避免了一些不必要重复，提高了程序效率。

【例 5.8】 在循环体中 break 语句执行示例。

```
#include <stdio.h>
void main(void)
{
   int i,sum=0;
   for(i=1;i<=5;i++)
   { sum+=i;
     if(sum>5) break;
     printf("sum=%d\n",sum);
   }
}
```

程序运行结果，如图 5-11 所示。

图 5-11　例 5.8 程序运行结果

本例中，如果没有 break 语句，程序将进行 5 次循环，输出 5 个结果，当 i=3 时，sum 的值为 6，if 的条件为“真”，从而执行 break 语句，跳出 for 循环，终止循环，输出两个结果。

5.6.2 continue 语句

continue 语句只能用在循环体中，其一般格式是：

continue;

其语义是：结束本次循环，即不再执行循环体中 continue 语句之后的语句，而是立即转入对循环条件的判断与执行。应特别注意的是，continue 语句只结束本层本次的循环，并不跳出循环。具体说：

(1) 对 while 和 do while 语句，continue 语句立即执行对 while 括号中条件表达式的判断；

(2) 对 for 语句，continue 语句在不执行其后的语句后，先求解“表达式 3”，再求解“表达式 2”。

【例 5.9】 在循环体中 continue 语句的使用。

```
#include <stdio.h>
void main(void)
{ int i;
   for(i=10;i<=50;i++)
   { if (i%5!=0)
       continue;
     printf("%5d",i);
   }
}
```

程序运行结果，如图 5-12 所示。

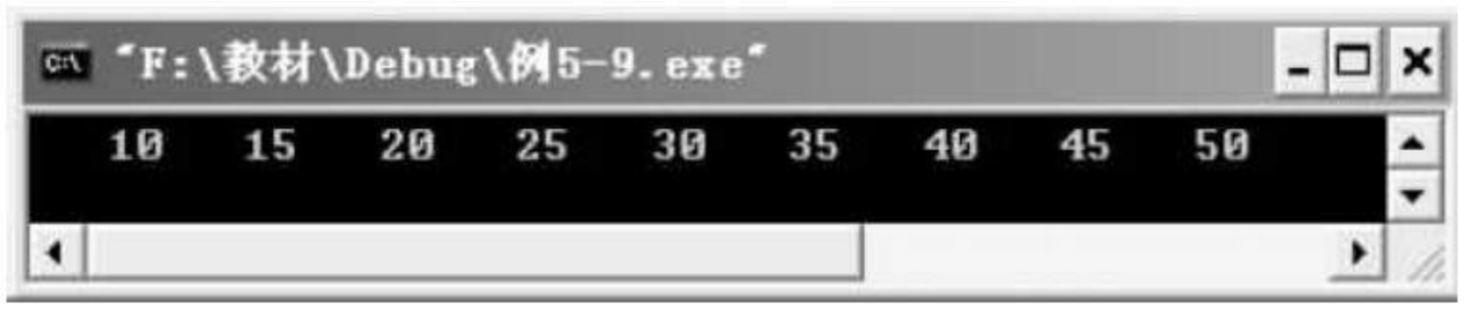


图 5-12 例 5.9 程序运行结果

该程序中当 i 不能被 5 整除时，执行 continue 语句，结束本次循环，只有 i 能被 5 整除时，才执行 printf 函数输出该数。

5.6.3 goto 语句与标号

在结构化程序设计中一般不主张使用 goto 语句，以免造成程序流程的混乱。能避免使用 goto 语句则应避免使用，当确实不得不用时，才使用 goto 语句。

goto 语句也称为无条件转移语句，其一般形式：

```
lable:if(e)
{
```

```
    goto lable;
}
```

goto 语句的语义是改变程序流向，立即跳转到标号所标识的语句。

语句标号是按标识符规定书写的符号，放在某一语句行的前面，标号后加冒号(:)。语句标号起标识语句的作用，并不影响该语句的执行，标号只起与 goto 语句配合的作用。

C 语言不限制程序中使用标号的次数，但各标号不得重名。

goto 语句通常与条件语句配合使用。可用来实现条件转移、构成循环、跳出循环体等功能。

使用 goto 语句时应该注意以下几点：

(1)跳转到一个循环内是非常危险的，应该极力避免这样做；

(2)不能跳转到本函数外；

(3)goto 语句越少用越好；

(4)通常不主张向程序的前面跳转。

【例 5.10】 使用 goto 语句求 1＋2＋3＋…100 的值。

```
#include <stdio.h>
void main(void)
{ int  i,sum=0;
  i=1;
lable:if(i<=100)
      { sum+=i;
        i++;
        goto lable;
      }
      printf("sum=%d\n",sum);
}
```

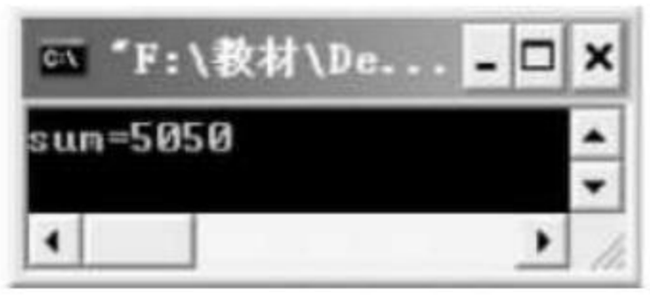



图 5-13　例 5.10 程序运行结果

程序运行结果，如图 5-13 所示。

该程序利用一个条件语句当“i<=100”为真值时执行花括号内的语句，其中有 goto 语句实现循环，从而求出 1 到 100 的和。

5.7　例题精解

【例 5.11】 用 $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 公式求 π 的近似值，直到某一项的绝对值小于 10^{-6} 为止。

```
#include <stdio.h>
#include <math.h>
void main(void)
{
   int s=1;
```

```
double n=1.0,t=1,pi=0;
while(fabs(t)>1e-6)
{ pi=pi+t;
  n+=2;
  s=-s;
  t=s/n;
}
pi=pi*4;
printf("pi=%f\n",pi);
}
```

图 5-14 例 5.11 程序运行结果

程序运行结果,如图 5-14 所示。

【解析】 本例利用多项式求出 π 的近似解。通过公式可见,求和的每一项分母都比前一项分母多 2,因此在循环中用 n+=2 实现求每项的分母;又每项数的符号正负交替,所以在循环中用 s=-s 实现,第一次是正数,第二次就为负数,第三次又为正数依次类推;用 t=s/n 表示和中的每一项,用函数 fabs(t)>1e-6 作为执行 while 循环的条件,满足题目要求。另外对 n 赋初值为 1.0 很重要,因为 n 作为分母实现的是整除运算,根据在运算中数的类型转换,若 n 赋初值为 1,结果取整为 0 就不正确了。

【例 5.12】 求输入的某个数是否为素数。若是,输出 yes,若不是,输出 no。

```
#include<stdio.h>
#include <math.h>
void main(void)
{ int i,n,flag=1;
  printf("Enter integer number:");
  scanf("%d",&n);
  i=2;
  while (i<n)
  {
    if(n%i==0)
    { flag=0;
      break;}
    else
      i++;
  }
  if (flag)
    printf("yes\n");
  else
    printf("no\n");
}
```

程序运行结果,如图 5-15 所示。

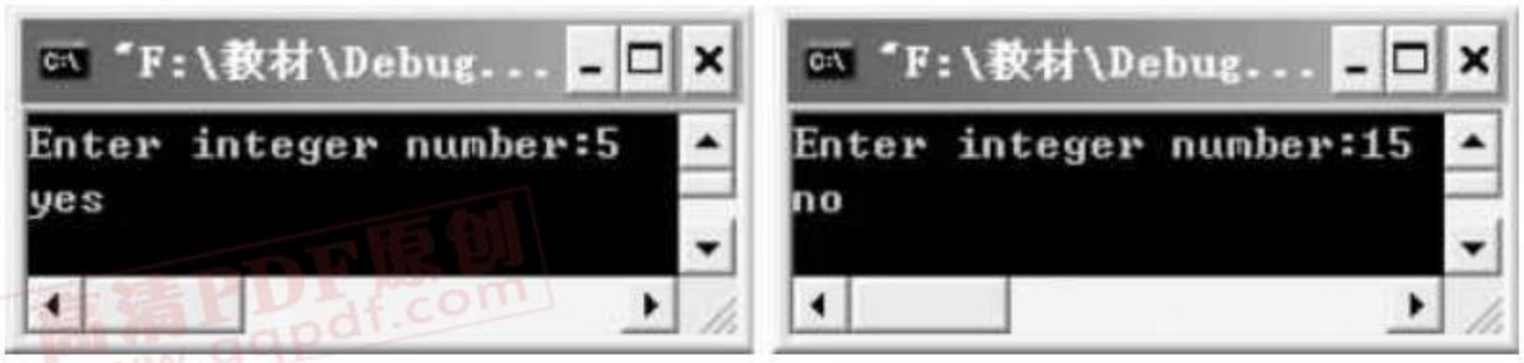


图 5-15　例 5.12 程序运行结果

【解析】　素数是指只能被 1 和它本身整除的数，如：2，3，5，7，11，…为素数。

对于自然数 n，判断其是否为素数有三种方法：

① 判断 n 是否能被从 2 到 n－1 范围内的数整除；

② 判断 n 是否能被从 2 到(int)(n/2)范围的数整除；

③ 判断 n 是否能被从 2 到(int)sgrt(n)范围的数整除。

为了判断数 n 是否为素数，最简单的方法是用 2 到 n－1 之间的数分别去除 n，只要有一个数能除尽，就不是素数，否则为素数。在这里 1 不是素数要单独考虑，程序中用一个变量 flag 作为标记，初值为 1，在运行过程中若 n%i＝＝0 为真值时，把该变量赋为 0，同时执行 break 终止循环，可以提高程序的效率，最后通过对 flag 的判断，确定该数是否为素数。

【例 5.13】　斐波那契数列的前几项为：1、1、2、3、5、8、13、21、……。编程输出该数列的前 n(n 由键盘输入)项，每行输出 5 个数。

```
#include <stdio.h>
void main(void)
{
   long i,f1=1,f2=1,f,n;
   printf("请输入 n:");
   scanf("%ld",&n);
   printf("%8ld%8ld",f1,f2);          /*输出前 2 项*/
   for(i=3;i<=n;i++)
   { f=f1+f2;
     printf("%8ld",f);
     if(i%5==0) printf("\n");         /*输出 5 项后换行*/
     f1=f2;f2=f;
   }
}
```

程序运行结果，如图 5-16 所示。

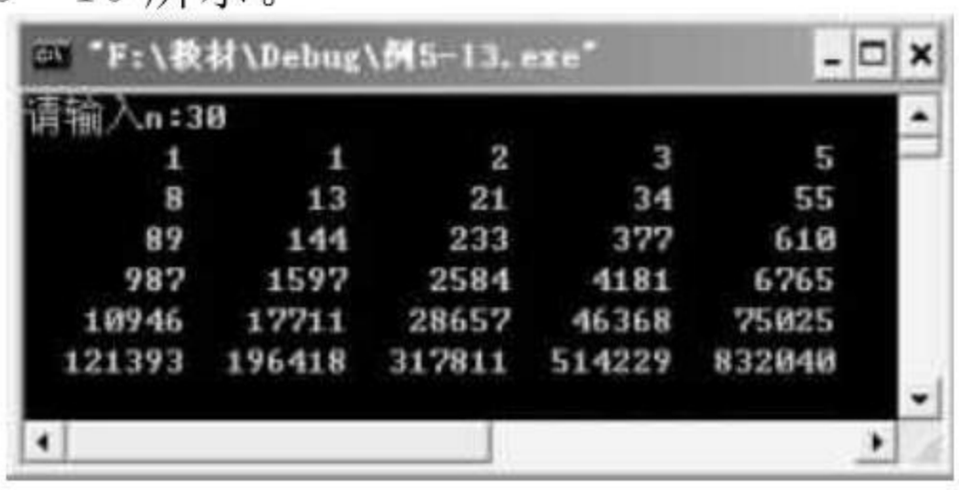


图 5-16　例 5.13 程序运行结果

【解析】 由题目给出的数列可以看到，数列的变化规律是：前两项都为1，从第三项开始，该数是前两项之和，因此可以用递推算法求出数列中每项。本程序中用了三个变量f1、f2、f。f1和f2的初始值为数列的前两项值为1，执行一次for循环求出后一项的值f=f1+f2，然后更新f1和f2的值，这样一次次重复执行。需要输出的项数由n确定。

【例5.14】 用辗转相除法(即欧几里德算法)求两个正整数的最大公约数。

```
#include<stdio.h>
void main(void)
{
  int m,n,q,a,b;
  printf ("Enter two integers:");
      scanf("%d,%d",&a,&b);              /* 输入两个正整数 */
  m=a;n=b;
  if(n>m)
  {   int z;
      z=m;m=n;n=z;
  }
  /*开始迭代*/
  do{
        q=m%n;   /*循环作整除取余，直到余数为0，则该被除数即为两数的最大公约数*/
        m=n;
        n=q;
     }while(q!=0);
    printf ("The greatest common divisor of");
    printf("%d,%d is %d\n",a,b,m);
}
```

程序运行结果，如图5-17所示。

图5-17 例5.14程序运行结果

【解析】 这是一个典型的迭代算法案例。设两个数m、n，假设n>=m，用n除以m，求得余数q，若q为0，则m即为最大公约数；若q不等于0，则如下迭代：m=n，n=q，原除数变为新的被除数，原余数变为新的除数重复除法，直至余数为0为止，余数为0时的除数n，即为原始m，n的最大公约数。

总结迭代的三个要素：

① 迭代初值：m、n的原始值。

② 迭代过程：q＝m％n；m＝n； n＝q；

③ 迭代条件：n!＝0

【例 5.15】 打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位数，其各位数字立方和等于该数本身。例如，153 是一水仙花数，因为 $153=1^3+5^3+3^3$。求水仙花数（条件：三位数的个、十、百位的方和等于该数。153＝＝13＋53＋33）。

```
#include <stdio.h>
void main(void)
{
   int n,a,b,c;
   for(n=100;n<=999;n++)      /*枚举所有三位数*/
   {
      /*取 n 的百位 a、十位 b、个位 c*/
      a=n/100;
      b=n%100/10;
      c=n%10;
      if(a*a*a+b*b*b+c*c*c==n)  /*测试条件*/
         printf("%d\n",n);
   }
}
```

程序运行结果，如图 5－18 所示。

图 5－18　例 5.15 程序运行结果

【解析】 水仙花数的求解，是一个典型的枚举算法案例。

程序中 n 为枚举变量，枚举初值为 100，枚举终值为 999；a 变量为 n 中的百位数、b 变量为 n 中的十位数、c 变量为 n 中的个位数；测试条件为三位数的个、十、百位的立方和等于该数，例如 $153==1^3+5^3+3^3$，满足条件输出 n 值。

本案例枚举的范围是 3 到 100 之间的奇数，测试条件可构造一个控制变量 i 从 $2\sim\sqrt{m}$ 的循环，让 m 被 i 逐一除，如果有一个能整除，则不是素数，退出测试。此时 i 的值小于等于 $\sqrt{m}$，如果都不能整除，同过循环正常退出时，i 的值$>\sqrt{m}$，m 即为素数。

5.8 本章小结

C语言中能够实现循环的语句主要包括：while语句、do－while语句、for语句。

while与do－while语句都是用于条件判断，当条件满足时（非0）执行循环体，否则退出循环。while与do－while结构的惟一区别在于：while结构循环是先判断条件，后执行循环体；而do－while结构循环是先执行循环体，后判断条件。通常循环次数及控制条件要在循环过程中才能确定的循环可用while或do－while语句。

for语句是最常用的循环语句，主要用于给定循环变量初值，步长增量以及循环次数的循环结构。功能与while相同，所不同的是：把控制变量的步长值放在括号内，显得更简洁，使用更方便。在使用循环结构时，要注意：

（1）三种循环语句可以相互嵌套组成多重循环。循环之间可以并列但不能交叉。

（2）可用转移语句把流程转出循环体外，但不能从外面转向循环体内。

（3）在循环程序中应避免出现死循环，即应保证循环变量的值在运行过程中可以得到修改，并使循环条件逐步变为假，从而结束循环。

习　　题

一、选择题

1. 以下程序段_____。

```
x=-1;
do
{x=x*x;}
while(!x);
```

A）是死循环　　B）循环执行二次

C）循环执行一次　　D）有语法错误

2. 下面是关于for循环的正确描述是_____。

A）for循环只能用于循环次数已经确定的情况

B）for循环是先执行循环体语句，后判断表达式

C）在for循环中，不能用break语句跳出循环体

D）for循环的循环体语句中，可以包含多条语句，但必须用花括号括起来

3. 设有以下程序段

```
#include <stdio.h>
void main(void)
{
   int x=0,s=0;
   while(!x!=0) s+=++x;
   printf("%d\n",s);
}
```

则_____。

A）运行程序段后输出 0　　B）运行程序段后输出 1

C）程序段中的控制表达式是非法的　　D）程序段执行无限次

4. 下面程序段的运行结果是_____。

```
#include <stdio.h>
void main(void)
{
  int x=0,y=0;
  while(x<15) y++,x+=++y;
  printf("%d,%d\n",y,x);
}
```

A) 20,7　　B) 6,12　　C) 20,8　　D) 8,20

5. 以下程序段的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int i,j,m=0;
  for(i=1;i<=15;i+=4)
    for(j=3;j<=19;j+=4) m++;
  printf("%d\n",m);
}
```

A) 12　　B) 15　　C) 20　　D) 25

6. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int y=10;
  for(;y>0;y--)
    if(y%3==0)
    { printf("%d",--y);continue;}
}
```

A) 741　　B) 852　　C) 963　　D) 875421

7. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int i;
  for(i=1;i<=5;i++)
  { if(i%2) printf("*");
    else continue;
    printf("#");
  }
```

```
    printf("$\n");
}
```

A) * # * # * # $　　B) # * # * # * $

C) * # * # $　　D) # * # * $

8. 下面程序的运行结果是_____。

```
#include<stdio.h>
void main(void)
{ int x,i;
  for(i=1;i<=100;i++)
  { x=i;
    if (++x%2==0)
      if (++x%3==0)
        if (++x%7==0)
          printf("%d\t",x);
  }
}
```

A) 39　81　　B) 42　84　　C) 26　68　　D) 28　70

9. 下列程序段不是死循环的是_____。

A)
```
int i=100;
while(1)
{ i=i%100+1;
  if(i>100) break;
}
```

B)
```
for(;;);
```

C)
```
int k=0;
do{++k;} while(k>=0);
```

D)
```
int s=36;
while(s); --s;
```

10. 若 x 是 int 型变量,以下程序段的输出结果是_____ 。

```
for(x=3;x<6;x++)
  printf((x%2)? ("* *%d\n") : ("# #%d\n"),x);
```

A)
```
* *3
 # #4
* *5
```

B)
```
# #3
 * *4
# #5
```

C)
```
# #3
* *4
 # #5
```

D)
```
* *3
# #4
 * *5
```

11. 下面程序的运行结果是_____。

```
#include <stdio.h>
void main(void)
{ int k=0;char c='A';
  do
  { switch(c++)
    { case 'A':k++;break;
      case 'B':k--;
```

```
        case 'C':k+=2;break;
        case 'D':k=k%2;continue;
        case 'E':k=k*10;break;
        default:k=k/3;
    }
    k++;
}
```



```
    while(c<'G');
    printf("k=%d\n",k);
}
```

A)k=3　　B)k=4　　C)k=2　　D)k=0

12. 下面程序的运行结果是_____。

```
#include <stdio.h>
void main(void)
{ int i,b,k=0;
  for( i=1;i<=5;i++)
  { b=i%2;
    while(b--<=0) k++;
  }
  printf("%d,%d\n",k,b);
}
```

A) 3,-1　　B) 8,-1　　C) 3,0　　D) 8,-2

13. 下面程序的运行结果是_____。

```
#include <stdio.h>
void main(void)
{ int i,j,a=0;
  for(i=0;i<2;i++)
  { for(j=0;j<4;j++)
    { if(j%2)break;
      a++;
    }
    a++;
  }
  printf("%d\n",a);
}
```

A) 4　　B) 5　　C) 6　　D) 7

二、填空题

1. 当执行以下程序段后,i 的值是_____、j 的值是_____、k 的值是_____。

```
#include <stdio.h>
```

```
void main(void)
{ int a,b,c,d,i,j,k;
   a=10; b=c=d=5;i=j=k=0;
   for( ;a>b;++b) i++;
   while(a>++c) j++;
   do k++; while(a>d++);
   printf("%d,%d,%d\n",i,j,k);
}
```

2. 用公式 $\frac{\pi^2}{6} \approx \frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\cdots\frac{1}{n^2}$ 求 π 的近似值，直到最后一项的值小于 10^{-6} 为止，请填空。

```
#include <stdio.h>
#include<math.h>
void main(void)                                    (1)_____
{ long i=1;
   __(1)__ pi=0;
   while(i*i>1e-6){pi=____(2)____;i++;}            (2)_____
   pi=sqrt(6.0*pi);
   printf("pi=%10.6f\n",pi);
}
```

3. 从键盘上输入若干学生的成绩，统计并输出最高成绩和最低成绩，当输入负数时结束输入，请填空。

```
#include <stdio.h>
void main(void)
{ float x,amax,amin;
   scanf("%f",&x);
   amax=x;amin=x;
   while(____(1)____)                              (1)_____
   { if(x>amax) amax=x;
     if(____(2)____) amin=x;                       (2)_____
     scanf("%f",&x);
   }
   printf("\namax=%f\namin=%f\n",amax,amin);
}
```

4. 输出 1 至 100 之间每位数乘积大于每位数的和的数，请填空。

```
#include <stdio.h>
void main(void)
{ int n,k=1,s=0,m;
   for(n=1;n<=100;n++)
```

```
{ k=1;s=0;
    (1)    ;                                    (1)_____
  while(    (2)    )
  { k*=m%10;                                    (2)_____
    s+=m%10;
    (3)    ;                                    (3)_____
  }
    if(k>s) printf("%d",n);
  }
}
```

5. 若用0至9之间不同的三个数构成一个三位数，下面程序将统计出共有多少种方法，请填空。

```
#include <stdio.h>
void main(void)
{ int i,j,k,count=0;
  for(i=1;i<=9;i++)
    for(j=0;j<=9;j++)
      if(  (1)  ) continue;                     (1)_____
      else
        for(k=0;k<=9;k++)
          if(    (2)    ) count++;               (2)_____
  printf("%d\n",count);
}
```

6. 有以下程序段：

```
s=1.0;
for(k=1;k<=n;k++) s=s+1.0/(k*(k+1));
printf("%f\n",s);
```

请填空，使下面的程序段的功能完全与之等同。

```
s=0.0;
    (1)    ;                                    (1)_____
k=0;
do
{ s=s+d;
    (2)    ;                                    (2)_____
  d=1.0/(k*(k+1));
} while(    (3)    );                          (3)_____
printf("%f\n",s);
```

三、应用题

1. 分析程序，写出程序运行结果。

```
#include <stdio.h>
void main(void)
{ int i,j,x=0;
  for(i=0;i<2;i++)
  { x++;
    for(j=0;j<=3;j++)
    { if(j%2) continue;
      x++;
    }
    x++;
  }
  printf("x=%d\n",x);
}
```

2. 分析程序,写出程序运行结果。

```
#include <stdio.h>
void main(void)
{ int i=5;
  do
  { switch(i%2)
    { case 0:i--;break;
      case 1:i--;continue;
    }
    i--;i--;
    printf("%d\n",i);
  }
  while(i>0);
}
```

3. 分析程序,写出程序的运行结果。

```
#include <stdio.h>
void main(void)
{ int i,k=19;
  while(i=k-1)
  { k-=3;
    if(k%5==0){i++;continue;}
    else if(k<5)break;
    i++;
  }
  printf("i=%d,k=%d\n",i,k);
}
```

4. 分析程序，写出程序的运行结果。

```
#include <stdio.h>
void main(void)
{ int a,y;
   a=10;y=0;
   do
   { a+=2;y+=a;
     if(y>50)break;
   } while(a<=14);
   printf ("a=%d,y=%d\n",a,y);
}
```


四、编写程序题

1. 编写程序，求 $1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{3^2}+\cdots+\frac{1}{15^2}$ 的值。

2. 编写程序，求 2－4＋6－8＋10－…－98＋100 的值。

3. 输入两个正整数 m 和 n，求其最大公约数和最小公倍数。

4. 求 S＝a＋aa＋aaa＋…＋aaa……a。

其中：0＜a＜10 共 n 项，最后一项有 n 个 a。

例如：a 为 2，S＝2＋22＋222＋…＋222…222(a 和 n 值由键盘输入)。

5. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1，2，3。而 6＝1＋2＋3，因此 6 是“完数”，编程序找出 1000 之内的所有完数。

6. 用循环语句输出九九乘法表。

第6章 数 组

【教学提示】

在现实世界中，对数据的处理要求千变万化，所以C语言除了前面介绍的基本类型数据外，还提供了构造类型数据：数组、结构体、共用体。所谓数组就是具有相同数据类型变量的有序集合。组成数组元素的变量(简称数组元素)，不同元素由其在数组中的序号即下标(从0开始编号)来标识。本章要求掌握数组的概念、定义和数组元素的引用方法，掌握一维数组、二维数组以及字符数组在程序设计中的应用。

【核心概念】

一维数组 数组元素 二维数组 字符串 字符串结束标志

6.1 一维数组

当只有一个下标的数组元素所组成的数组通常称为一维数组。

6.1.1 一维数组的定义

数组与变量相同，必须先定义，后使用。

【例6.1】 实例。

```
#include <stdio.h>
void main(void)
{ int i;
  int a[10];                    /*定义一个一维数组 a */
  for(i=0;i<10;i++)             /*通过循环对数组 a 的 10 个元素赋值*/
    a[i]=2*i;                   /*数组的引用*/
  for(i=0;i<10;i++)             /*通过循环输出 10 个元素的值 */
    printf("%4d", a[i]);        /*数组的引用*/
}
```

程序运行结果，如图6-1所示。

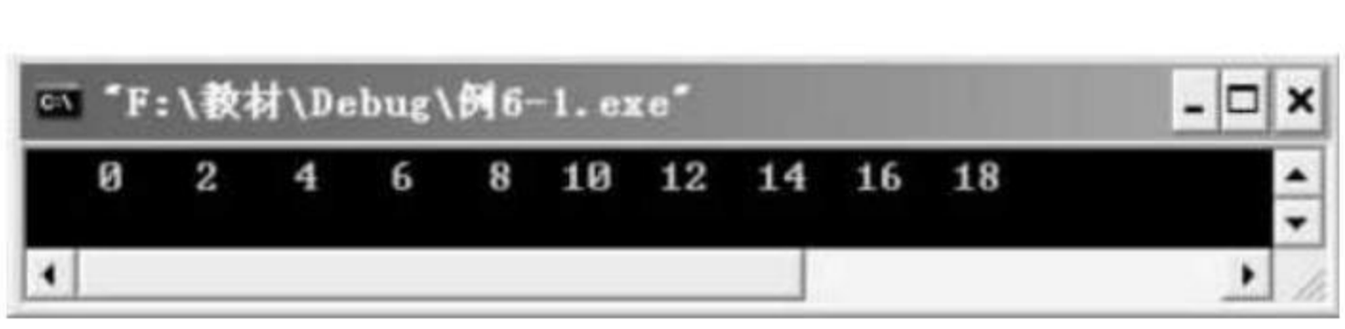


图6-1 例6.1程序运行结果

在上述的程序中，我们可以看出，一维数组定义的一般形式为：

类型说明符 数组名[常量表达式];

例如：int a[10];

几点说明：

(1)数组名是标识符,所以数组命名要遵守标识符命名规则,并且不能与其他变量同名。

(2)常量表达式就是数组元素的个数,即数组的长度。如:上述的数组a有10个元素,分别是a[0]、a[1]、…、a[9]。绝对不可以使用变量或含有变量的表达式。

(3)定义数组时,系统会在内存中分配一段连续的空间,各元素将按顺序占有这段内存,如图6-2所示。各元素所占的内存空间与数组类型有关,上述的数组a类型为int,所以每个元素占2个字节。

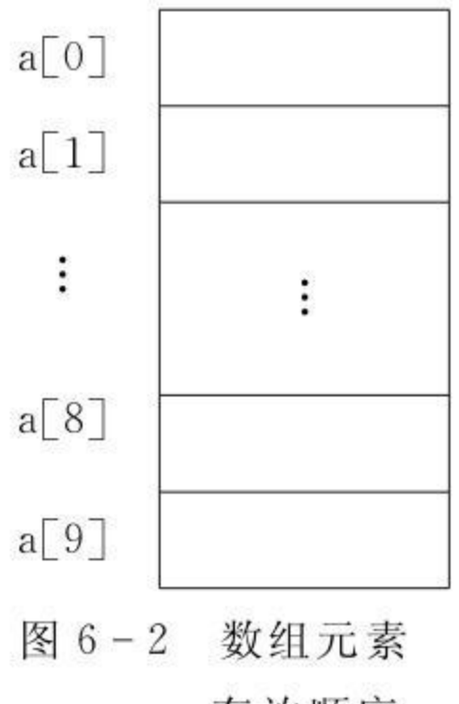


图6-2　数组元素存放顺序

与变量定义相同,类型说明符前可以加上“存储类别”,如:static、extern等。与变量相同,数组定义后,在没有赋值前每个元素的值是不确定的。

注意:在定义数组后,系统自动为文分配一块连续的内存空间。此时,数组名指向这块空间的起始点,并且在程序中不可改变。

6.1.2　一维数组元素的引用

定义数组目的是为了引用,在C语言中,数组元素只能逐个单独引用,而不能整体引用数组元素。

在**【例6.1】**中有两处对数组元素进行引用,可以看出,一维数组元素的引用形式为:

数组名[下标];

下标可以是常量或有确定值的变量、表达式。

【例6.2】　分析下面程序的运行结果。

```
#include <stdio.h>
void main(void)
{ int i;
  int a[10];
  for(i=0;i<5;i++)
  { a[2*i]=i;                    /*数组元素的引用*/
    a[2*i+1]=a[2*i];             /*数组元素的引用*/
  }
  for(i=0;i<10;i++)
    printf("%4d", a[i]);         /*数组元素的引用*/
}
```

程序运行结果,如图6-3所示。

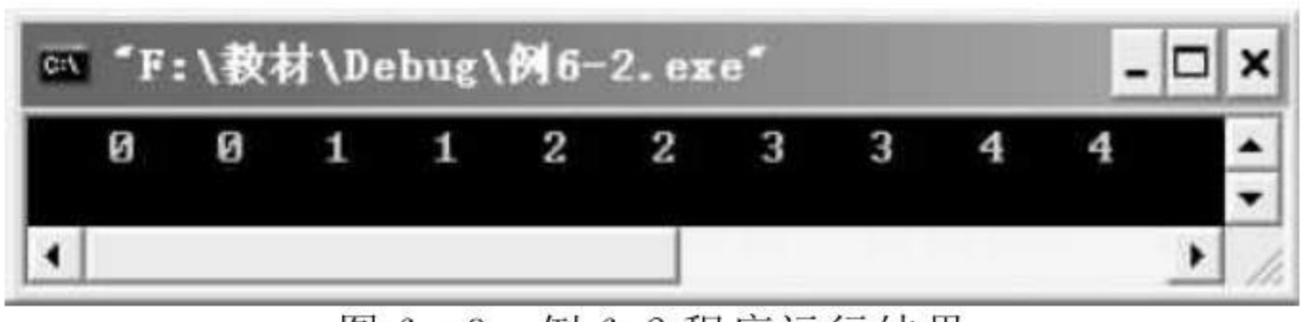


图6-3　例6.2程序运行结果

6.1.3 一维数组的初始化

数组的初始化就是在定义数组的同时对数组元素赋值。例如，使用初始化将【例 6.1】中的程序改写为：

```
#include <stdio.h>
void main(void)
{ int i;
  int a[10]={0,2,4,6,8,10,12,14,16,18};   /*通过初始化对 10 个元素赋值*/
  for(i=0;i<10;i++)                       /*通过循环输出 10 个元素的值*/
    printf("%4d", a[i]);                  /*数组的引用*/
}
```

一维数组的初始化有多种方法，具体有以下几种形式：

1. 对全部元素初始化，如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

经过初始化后，从 a[0]到 a[9]的值分别是从 0 到 9 的整数。

在对全部元素初始化时，可以省去数组长度，如：

```
int a[]={0,1,2,3,4,5,6,7,8,9};
```

这样的初始化与上面代数组长度的初始化完全等价。

2. 对部分元素初始化，如：

```
int a[10]={0,1,2,3,4};
```

经过初始化后，前 5 个元素的值分别是从 0 到 4，后 5 个元素的值为零。

6.1.4 一维数组的应用

【例 6.3】 现有 10 个员工的当月奖金，分别是 100，200.6，300，120，480.2，20，310，80.5，90，210，编程求出最高奖金与平均奖。

```
#include <stdio.h>
void main(void)
{ float a[10]={ 100,200.6,300,120,480.2,20,310,80.5,90,210 };
  float max,average,sum;
  int i;
  max=a[0];sum=0;
  for(i=1;i<10;i++)
  { if (a[i]>max)
      max=a[i];
    sum=sum + a[i];
  }
  average=sum/10;
  printf("max=%6.2f,average=%6.2f",max,average);
}
```

程序运行结果，如图 6－4 所示。

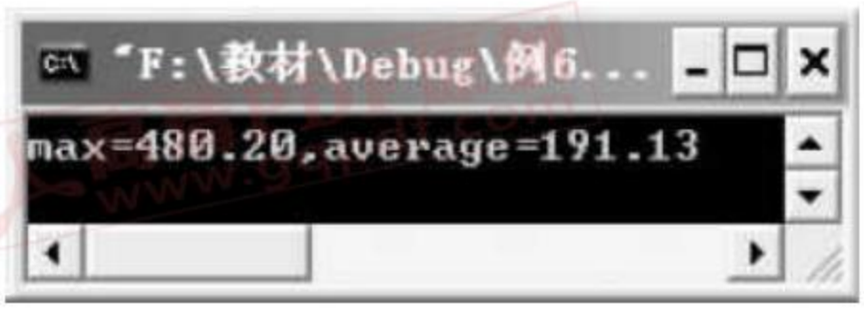


图 6－4　例 6.3 程序运行结果

【例 6.4】　用数组来处理求 Fibonacci 数列问题。Fibonacci 数列的前两项都为 1，以后各项是前两项之和，编程求出 Fibonacci 数列的前 20 项。

```
#include <stdio.h>
void main(void)
{
  int i;
  int f[20]={1,1};
  for(i=2;i<20;i++)
    f[i]= f[i-2]+f[i-1];          /* 当前项是前两项之和 */
  for(i=0;i<20;i++)
  { if(i%5==0) printf("\n");      /* 控制每输出 5 项换行一次 */
    printf("%10d", f[i]) ;
  }
}
```

程序运行结果，如图 6－5 所示。

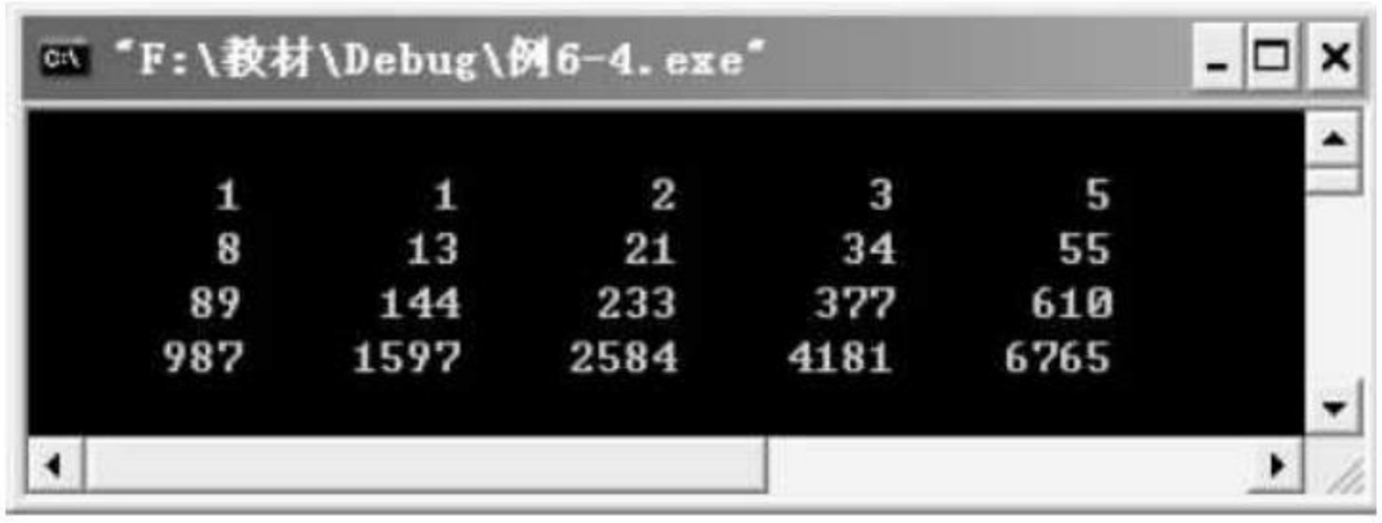


图 6－5　例 6.4 程序运行结果

6.2　二维数组

有两个下标的数组元素所组成的数组称为二维数组。C 语言允许使用多维数组，二维数组是多维数组中最简单的形式，应该重点掌握。

6.2.1　二维数组的定义

【例 6.5】　实例。

```
#include <stdio.h>
void main(void)
{
  int i,j,a[3][3];      /* 定义一个二维数组 a */
  for(i=0;1<3;i++)
```

```
    for(j=0;j<3;j++)
      a[i][j]=i+j;     /*通过两重循环对二维数组 a 的 9 个元素赋值*/
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
      printf("%4d",a[i][j]);      /*按 3 行 3 列形式,输出各元素的值*/
      }
}
```

程序运行结果,如图 6-6 所示。

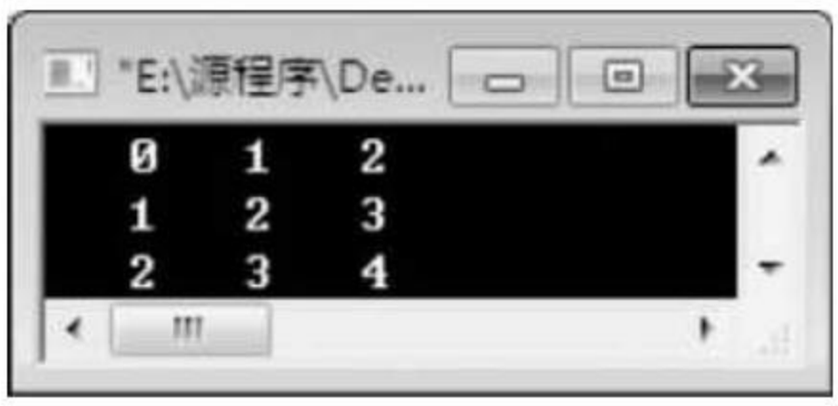


图 6-6　例 6.5 程序运行结果

在上述的程序中,我们可以看出,二维数组定义的一般形式为:

类型说明符 数组名[常量表达式 1][常量表达式 2];

例如:int a [3][3];

几点说明:

二维数组可以看为具有行和列的矩阵,对于上述的二维数组 a 是一个 3 行 3 列(3×3)的矩阵,共有 9 个元素,分别是:

	第 1 列	第 2 列	第 3 列
第 1 行	a[0][0]	a[0][1]	a[0][2]
第 2 行	a[1][0]	a[1][1]	a[1][2]
第 3 行	a[2][0]	a[2][1]	a[2][2]

二维数组可以看作是一种特殊的一维数组,对于上述的二维数组 a,可以把 a 看作是一个一维数组,它含有 3 个元素:a[0]、a[1]、a[2],每个元素又都是含有 3 个元素的一维数组,数组名分别是 a[0]、a[1]、a[2],如图 6-7 所示。

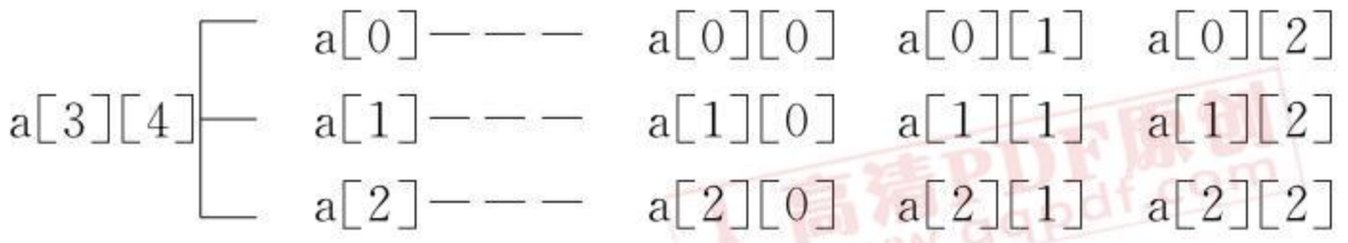


图 6-7　二维数组的排列顺序

二维数组中元素排列的顺序是:按行存放,即在内存中先顺序存放第一行的元素,再存放第二行的元素,如图 6-8 所示。

6.2.2　二维数组元素的引用

二维数组元素引用形式为：

数组名[下标][下标]；

其中，下标为整型常量或整型表达式。

例如，在【例 6.5】的程序中：

```
a[i][j]=i+j;
printf("%4d",a[i][j]);
```

这两处均为二维数组元素的引用。

几点说明：

(1)与一维数组一样，二维数组元素可以像普通数据一样进行赋值、运算、输入及输出操作。

(2)二维数组的两个下标可以是常量或有确定值的变量、表达式。

(3)在引用数组元素时，要注意下标的值应该在已定义的数组的范围内，虽然数组越界没有语法错误，但可能会破坏数据，比较危险。

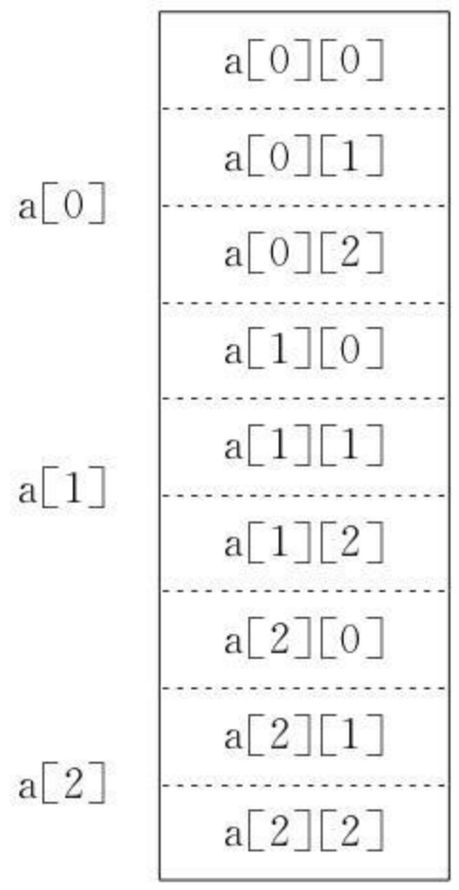


图 6-8　二维数组存放顺序

6.2.3　二维数组的初始化

二维数组的初始化，有以下几种方式。

1. 按行对元素赋值

(1) 对所有元素初始化

```
int  a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

(2) 对部分元素初始化

```
int a[3][4]={{1},{2,3},{4,5,6}};
```

没有赋值的元素为零，所以，a[0][0]、a[0][1]、…、a[2][3]的值分别是：1,0,0,0,2,3,0,0,4,5,6,0。

2. 按顺序对元素赋值

将所有元素的值写在一个花括号内，按数组的排列顺序对所有元素赋值或对部分元素赋值。

(1) 对所有元素初始化

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

(2) 对部分元素初始化

```
int a[3][4]={1,2,3,4,5,6};
```

没有赋值的元素为零，所以，a[0][0]、a[0][1]、…、a[2][3]的值分别是：1,2,3,4,5,6,0,0,0,0,0,0。

3. 缺省第一维下标

(1) 对所有元素初始化

```
int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

或:int a[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

(2) 对部分元素初始化

int a[][4]={{1},{2,3},{4,5,6}};

注意:对部分元素初始化,必须使用按行赋值的办法。

6.2.4 二维数组的应用

【例 6.6】 将一个二维数组行与列互换,存到另一个二维数组中。

$$a=\begin{vmatrix}1 & 2 & 3\\4 & 5 & 6\end{vmatrix} \qquad b=\begin{vmatrix}1 & 4\\2 & 5\\3 & 6\end{vmatrix}$$

```
#include <stdio.h>
void main(void)
{ int a[2][3]={{1,2,3},{4,5,6}},b[3][2],i,j;
   printf("数组 a 是:\n");
   for(i=0;i<2;i++)
   { for(j=0;j<3;j++)
     { printf("%4d",a[i][j]);
       b[j][i]= a[i][j];          /* 将 a 的 i 行 j 列元素值赋值给 b 的 j 行 i 列元素 */
     }
   printf("\n");                  /* 每输完一行进行换行 */
   }
   printf("数组 b 是:\n");
     for(i=0;i<3;i++)
     { for(j=0;j<2;j++)
         printf("%4d",b[i][j]);
       printf("\n");
     }
}
```

图 6-9 例 6.6 程序运行结果

程序运行结果,如图 6-9 所示。

【解析】 这是一个非方阵转置。程序中要分别定义两个数组:a 数组存放了原始数据,b 数组是转置后存放的数据。此程序实现转置的关键语句:b[j][i]=a[i][j]。

【例 6.7】 有如下 3×4 矩阵 a,编程求出最大的元素值及其所在的行号与列号。

$$a=\begin{vmatrix}5 & 12 & 20 & 21\\9 & 22 & 18 & 32\\4 & 1 & 16 & 27\end{vmatrix}$$

```
#include <stdio.h>
void main(void)
```

```
{ int i,j,row=0,colum=0,imax;
  int a[3][4]={{5,12,20,21},{9,22,18,32},{4,1,16,27}};
  imax=a[0][0];
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      if(a[i][j]>imax)
      { imax=a[i][j];
        row=i;
        colum=j;
      }
  printf("最大的元素值是:%d\n 行号为:%d,列号为:%d\n",imax,row,colum);
}
```

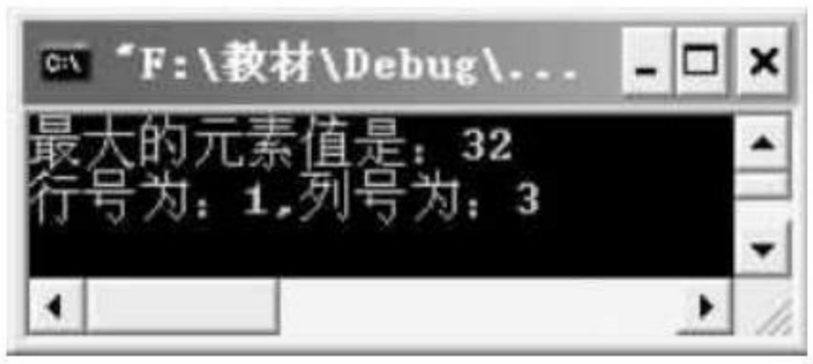


图 6-10　例 6.7 程序运行结果

程序运行结果,如图 6-10 所示。

【解析】 此程序是在 3×4 二维数组中查找最大值和最大值的行列位置号。首先把 a 数组的第 1 行第 1 列赋给最大值变量,再由最大值与其余元素相比较,最后找出最大值以及位置号。

6.3　字符数组和字符串

6.3.1　字符数组的定义

用来存放字符数据的数组是字符数组,字符数组中的一个元素存放一个字符。字符数组的定义方法与前面类似,例如:

```
char c[10];
```

6.3.2　字符数组的初始化

字符数组的初始化有两种方法:

1. 将字符写在花括号内,按顺序赋给数组中的元素,如:

```
char c[6]={'t','u','r','b','o','c'};
```

说明:

(1) 当花括号内的字符个数等于数组长度时,可以省去数组长度。如:

```
char c[]={'t','u','r','b','o','c'};
```

系统自动默认该数组长度为 6。

(2) 当花括号内的字符个数少于数组长度时,将这些字符赋给前面的元素,后面元素自动赋空字符,即'\0'。

如:char c[6]={'c','+','+'};

从 c[0]到以 c[5]的值分别是:'c'、'+'、'+'、'\0'、'\0'、'\0'。当花括号内的字符个数大于数组长度时,出现语法错误。

重点:当给出的初始值个数少于元素个数时,从首元素开始赋值,剩余元素默认值为空字符,即'\0'。

2. 以字符串格式进行初始化

C语言没有提供字符串数据类型,可以使用结尾元素是字符'\0'的字符数组来处理字符串,'\0'是字符串的结束标志。

【例6.8】 请写出下面程序的运行结果。

```
#include <stdio.h>
void main(void)
{ char c[9]={'T','u','r','b','o',' ','C','2','\0'};
  printf("%s\n",c);
  c[1]='\0';                /*使c[1]变成结束标志,其他元素值不变*/
  printf("%s",c);           /*将数组c存储的字符串输出。注意%s的使用*/
}
```

程序运行结果,如图6-11所示。

以字符串格式对字符数组进行初始化格式如下:

```
char c[9]="Turbo C2";
```

或:`char c[9]={"Turbo C2"};`

在【例6.8】中的对字符数组c进行初始化可以改写成:

```
char c[9]= "Turbo C2";
```

图6-11 例6.8程序运行结果

说明:

(1) 要考虑字符串的结束标志'\0'。若字符串的字符数为N个,字符数组的元素个数最少需要N+1个。

(2) 若字符数组的长度刚好满足字符串需要时,可以省去长度。例如:

```
char c[]="Turbo C2";
```

系统默认数组c的长度是9,最后一个字符是系统自动添加的结束标志'\0'。

可以用二维字符数组存储多个字符串。例如:

```
char c[3][10]={"Turbo C2","c++","c#"};
```

注意:当程序中用到系统提供的一些字符串函数时要包含头文件<string.h>。

6.3.3 字符数组的引用

字符数组的引用与前面数组引用相似,可以引用字符数组中的一个元素,得到一个字符。

【例6.9】 任意输入10个英文字母,用大写字母输出。

```
#include <stdio.h>
void main(void)
{ char c[10];
  int i;
  printf("请输入10个英文字母:\n");
  for(i=0;i<10;i++)
```

```
{ scanf("%c",&c[i]);
  if (c[i]>='a'&&c[i]<='z')
     c[i]=c[i]-32;
}
for(i=0;i<10;i++)
  printf("%c",c[i]);
}
```

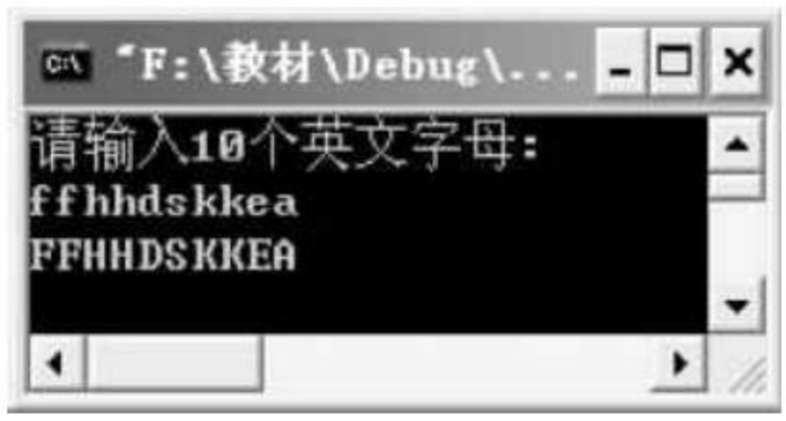


图 6-12　例 6.9 程序运行结果

程序运行结果，如图 6-12 所示。

6.3.4　字符数组的输入与输出

1. 逐个字符输入与输出

在 scanf 与 printf 函数中使用格式符“%c”，输入或输出一个字符，如【例 6.9】。

2. 将整个字符串一次输入或输出

将整个字符串一次输入输出，用%s 格式符，例如：

```
char c[6]="china";
printf("%s",c);
```

输出时，遇结束符"\0"就停止输出。

说明：

(1) 输入的字符串长度(字符个数)必须小于数组长度，系统自动在字符串最后一个字符后加上结束标志‘\0’。

(2) 输出字符串时，一个字符数组若存在多个‘\0’，第一个‘\0’才是结束标志，输出第一个‘\0’前面所有字符，不包括结束标志。

(3) 在 scanf 与 printf 函数中输出项是数组名，而不是元素名，例如：

```
#include<stdio.h>
void main(void)
{ char c[10];
  scanf("%s",c);
  printf ("%s",c);
}
```

(4) 在使用 scanf 函数输入多个字符串时，可以使用空格或回车作为字符串的分隔符。例如：

```
char str1[6],str2[6],str3[6];
scanf("%s%s%s", str1,str2,str3);
```

输入数据：

```
How are you! ↙
```

三个数组在内存的存放形式，如图 6-13 所示。

str1	H	o	w	\0		
str2	a	r	e	\0		
str3	y	o	u	!	\0	

图 6-13

3. 使用字符串处理函数对字符数组进行输入或输出见下一节。

6.3.5 字符串处理函数

C的函数库中提供了一些专门用来处理字符串的函数,常用的函数如下:

1. 字符串输出函数

一般形式为:

puts(字符数组)

作用:输出字符数组所存的字符串。

例如:char str[6]= "china";

puts(str);

输出:china。

注意:函数参数的“字符数组”应该是数组名,后面介绍的函数也是如此。

2. 字符串输入函数

一般形式为:

gets(字符数组)

作用:从键盘输入一个字符串,并将这个字符串保存到字符数组中。

例如:char str[6];

gets(str);

从键盘输入:china ↙

执行后字符数组 str 的在内存的存放形式,如图 6-14 所示。

c	h	i	n	a	\0

图 6-14 字符串存放形式

3. 字符串复制函数

一般形式为:

strcpy(字符数组 1,字符数组 2 或字符串常量)

作用:将字符串 2 复制到字符数组 1 中。

例如:char str1[6],str2[6];
strcpy(str2,"china");

strcpy(str1,str2);

执行后,str1 与 str2 的在内存的存放形式,如图 6-15 所示。

str1	c	h	i	n	a	\0
str2	c	h	i	n	a	\0

图 6-15 字符串存放形式

4. 求字符串长度函数

一般形式为:

strlen(字符数组)

作用:求出字符数组的长度,即所含字符的个数。例如:

char str[10]= "china";

int n;

n=strlen(str);

执行后,n 被赋值为 5。

6.3.6　字符数组的应用

【例6.10】　随意输入20个字符，统计大写字母和小写字母的个数，并将这20个字符逆序输出。

```
#include <stdio.h>
void main(void)
{ char c[20];
  int i,m=0,n=0;
  printf("请任意输入20个字符:\n");
  for(i=0;i<20;i++)
    { scanf("%c",&c[i]);
      if (c[i]>='a'&&c[i]<='z') m++;
      if(c[i]>='A'&&c[i]<='Z') n++;
    }
  printf("小写字母有%d个,大写字母有%d个",m,n);
  for(i=19;i>=0;i--)
    printf("%c",c[i]);
}
```

程序运行结果，如图6-16所示。

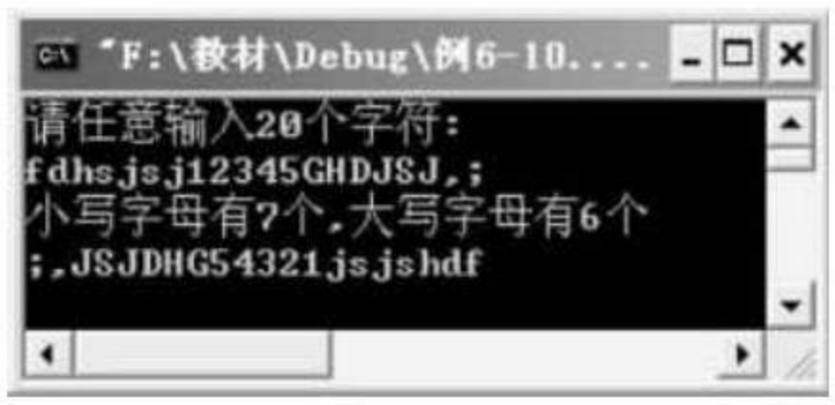


图6-16　例6.10程序运行结果

【例6.11】　输入两个字符串，将两字符串连接后输出。

```
#include <stdio.h>
#include<string.h>
void main(void)
{ char str1[80],str2[80];
  int i,j=0;
  printf("请输入第一个字符串:\n");
  gets(str1);
  printf("请输入第二个字符串:\n");
  gets(str2);
  i=strlen(str1);
  while(str2[j++]!='\0')
    str1[i++]=str2[j];
```

```
    str1[i]='\0';
    printf("连接后的字符串:\n");
    puts(str1);
}
```

程序运行结果,如图 6-17 所示。

图 6-17　例 6.11 程序运行结果

6.4　例题精解

【例 6.12】　冒泡法排序。随机输入 10 个整数,将它们按由小到大的顺序输出。

```
#include <stdio.h>
void main(void)
{ int a[10];
  int i,j,t;
  printf("请输入 10 个数字:\n");
  for(i=0;i<10;i++)
    scanf("%d",&a[i]);
  printf("\n");
  for(i=0;i<=8;i++)
    for (j=0;j<=9-i;i++)
      if (a[j]>a[j+1])
      { t=a[j];a[j]=a[j+1];a[j+1]=t;}
  printf("排序后结果是 :\n");
    for(i=0;i<10;i++)
      printf("%5d",a[i]);
}
```


程序运行结果,如图 6-18 所示。

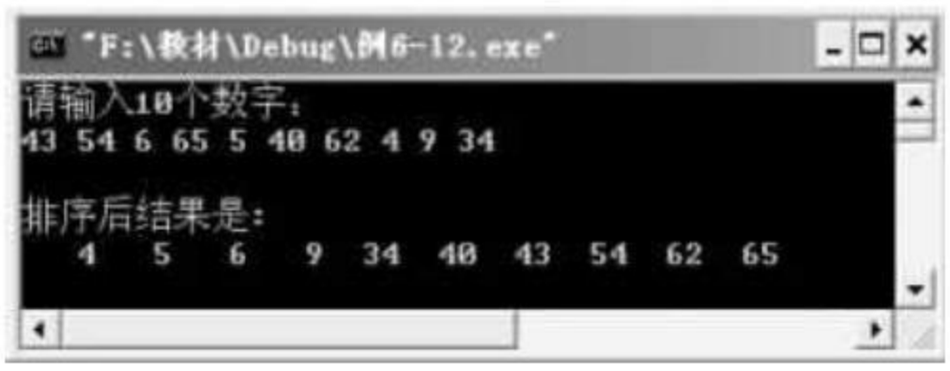
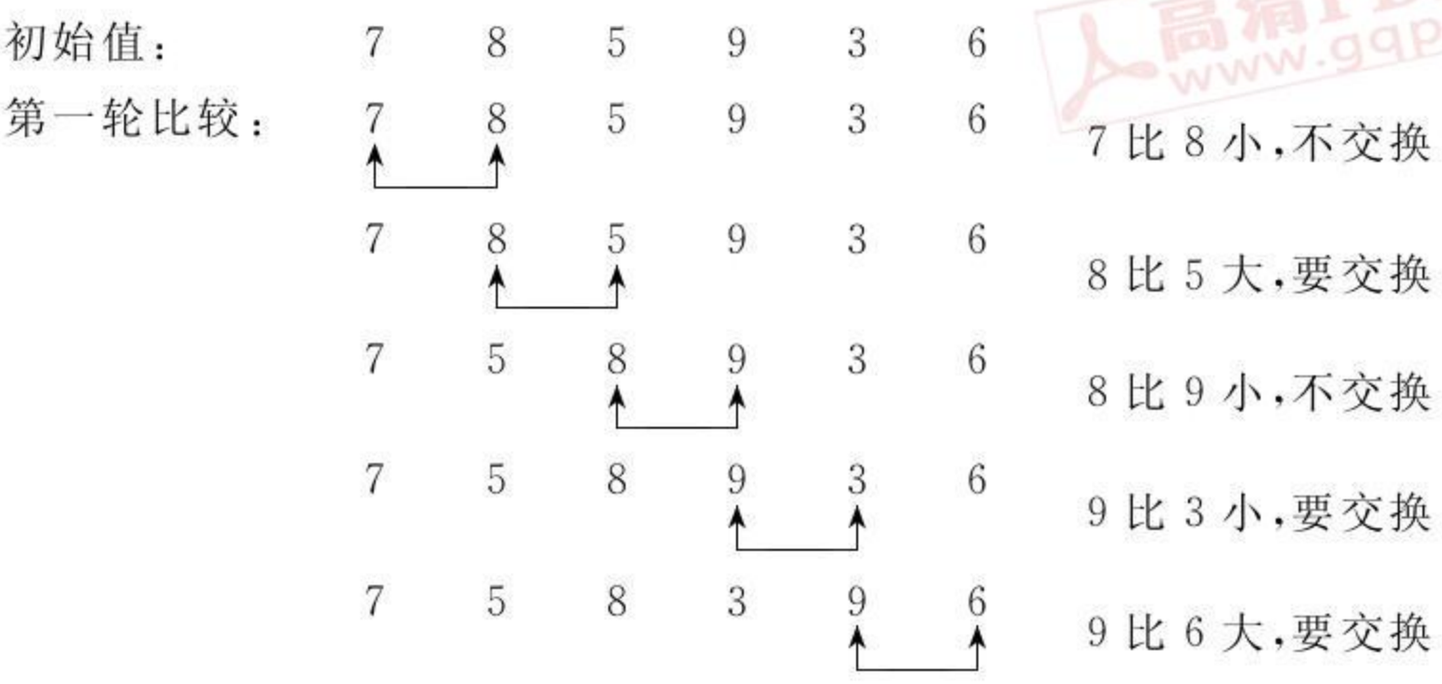


图 6-18　例 6.12 程序运行结果

【解析】 用冒泡法的思路是：将相邻两个数比较，将小的调到前头。假定有 6 个数（图 6-19），第一轮进行 5 次比较，将最大的数“沉”到最后一位。第二轮比较 4 次，将次大的数放到倒数第二位。如此进行下去，可以推知 6 个数要比较 5 轮。在第 1 轮中要进行两两比较 5 次，在第 2 轮中比 4 次，…，第 5 轮比 1 次。如果有 n 个数，则要进行 n—1 轮比较。



第 1 轮比较结果：7　5　8　3　6　9

第 2 轮比较结果：5　7　3　6　8　9

第 3 轮比较结果：5　3　6　7　8　9

第 4 轮比较结果：3　5　6　7　8　9

第 5 轮比较结果：3　5　6　7　8　9

图 6-19　冒泡排序法

【例 6.13】 选择法排序。

```
#include<stdio.h>
void main(void)
{
   int n,i,m,x,min,j,a[100];
   printf("输入 n 值:");
   scanf("%d",&n);                          /* 输入要排序数的个数,n 不能大于 100 */
   printf("\n 输入%d 个数:",n);
   for(i=0;i<n;i++)                         /* 输入 n 个要求排序的数 */
       scanf("%d",&a[i]);
   for(m=0;m<n-1;m++)                       /* 控制排序总共进行 n-1 步 */
     {
       min=a[m];                            /* 设定第 m 个数是当前最小数 */
       j=m;                                 /* 用 j 记录最小数的下标 */
       for(i=m;i<n;i++)                     /* 寻找最小数 */
           if(a[i]<min)
             {
                min=a[i];                   /* 记录新的最小数和下标 */
                j=i;
```

```
            }
        x=a[j];                                  /* 交换最小数和第 m 个数的位置 */
        a[j]=a[m];
        a[m]=x;
    }
    printf("\n 输出排序后的值:");
    for(i=0;i<n;i++)
        printf("%5d",a[i]);
    printf("\n");
}
```

程序运行结果,如图 6-20 所示。

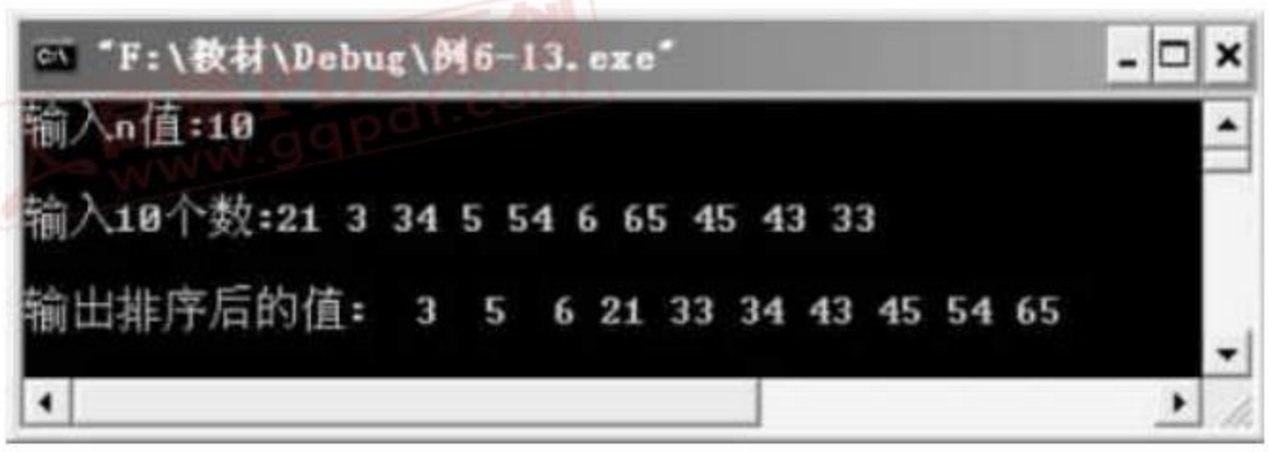


图 6-20 例 6.13 程序运行结果

【解析】 选择排序法是一种简单而且常用的对数据排序的方法,其排序原理如下:

设有 m 个数要求从小到大排列,选择排序法排序过程分为 m−1 个步骤:

第 1 步,在 m 个数中找出最小数,然后和第一个数交换,前 1 个数已经排好序。

第 2 步,在 m−1 个数中找出最小数,然后和第二个数交换,前 2 个数已经排好序。

……

第 k 步,在 m−k+1 个数中找出最小数,然后和第 k 个数交换,前 k 个数已经排好序。

这样,一直到 m−1 步结束。

【例 6.14】 二分法查找(又称对半查找)。

二分法查找的前提,只能在有序的数列中查找。

```
#include <stdio.h>
#define N 10
void main(void)
{ int k,i;
   int a[N]={1,3,5,7,9,11,13,15,17,19};
   int top=0,m,bot=N-1;
   printf("请输入要查找的数:\n");
   scanf("%d",&k);
   while(top<=bot)
      { m=(top+bot)/2;
        if(k==a[m])
```

```
        break;
    else if(k<a[m])
        bot=m-1;
    else top=m+1;
}
if(top>bot)
    printf("%d 不在这个数列中\n",k);
else
    printf("%d 在 a[%d]中\n",k,m);
}
```

程序运行结果,如图 6-21 所示。

图 6-21　例 6.14 程序运行结果

【解析】 将 10 个已排序好的整数:1,3,5,7,9,11,13,15,17,19,随便输入一个整数,查找该整数是否在这个序列中。

二分法查找的思路是:假定数据是按升序排列的,对于给定值 X,从序列的中间位置开始比较,如果当前值等于 X,则查找成功;否则若 X 小于当前值,则在序列的前半段数据中查找,若 X 大于当前值,则在序列的后半段数据中查找,直到找到给定的值 X,则查找成功;若表示序列查找范围的上、下界数值颠倒,查找不成功。

对于上述序列,假设查找定义值 X=5。查找过程如下:

用 top、bot 作为查找范围的上、下界,用 m 表示每次比较的数据的位置,m=(top+bot)/2。最初 top=0、bot=9,则 m=4,若用"[]"代表查找范围,带下划线的是要比较的当前数,即位置在 m 上的数,则有下列表示:[1,3,5,7,9,11,13,15,17,19]。

由于 X<9,所以应在前半段查找。重新设定查找范围:top=0,bot=m-1=3,m=1,则有下列表示:[1,3,5,7],9,11,13,15,17,19。由于 X>3,所以应在后段查找。重新设定查找范围:top=m+1=2,bot=3,m=2,则有下列表示:1,3,[5,7],9,11,13,15,17,19。

这时当前值为 5,与 X 相等,则查找成功。若要查找的数 K=6,则会出现上界值与下界值颠倒的情况,表示查找不成功。

【例 6.15】 有一个 3×3 矩阵,求 $a=\begin{vmatrix}10 & 11 & 12\\13 & 14 & 15\\16 & 17 & 18\end{vmatrix}$ 两对角线上元素和。

```
#include <stdio.h>
void main(void)
{ int a[3][3]={{10,11,12},{13,14,15},{16,17,18}};
```

```
int i,j,m1=0,m2=0;
for(i=0;i<3;i++)
  for(j=0;j<3;j++)
    { if (i==j)
        m1+=a[i][j];
      if (i+j==2)
        m2+=a[i][j];
    }
  printf("第一条对角线元素和是:%d,第二条对角线元素和是:%d",m1,m2);
}
```

程序运行结果,如图 6-22 所示。

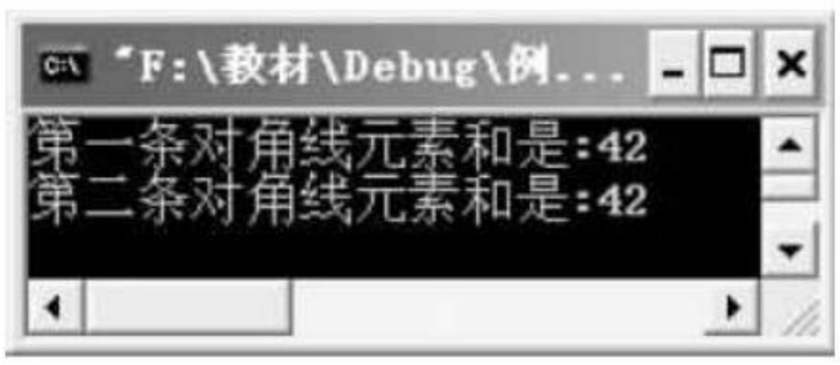


图 6-22　例 6.15 程序运行结果

【解析】 思路:第一条对角线的元素是 a[0][0]、a[1][1]、a[2][2],两角下标相等。第二条对角线的元素是 a[0][2]、a[1][1]、a[2][0],两角下标之和为 2。只要将下角标相等的元素相加即可得到第一条对角线上的元素和;将两角下标和为 2 的元素相加即可得到另一条对角线上的元素和。

6.5　本章小结

本章主要介绍了一维数组、二维数组及字符数组。数组是一个由若干同类型变量组成的有序集合,数组中特定的元素通过数组名和下标来访问。数组由连续存储单元存储,数组名本身是地址,对应于数组的第一个元素地址。

C 语言中数组的下标是从 0 开始,二维数组在内存中存储顺序是按行存储的。数组的赋值有三种方法:数组初始化赋值、输入函数赋值、赋值语句赋值。注意不能用赋值语句对数组整体赋值,必须用循环语句逐个对数组元素赋值,同样不能对数值型数组整体进行输入、输出。可以对字符数组元素进行赋值、比较,一个字符在内存中存储占一个字节。字符数组的实质就是一个字符串,在 C 语言中,用'\0'表示字符串结束,因此在用字符串给字符数组赋值时,数组的长度要比字符串长度多 1。

系统提供了一些字符串处理函数,用来为用户提供一些字符串的处理,在程序中可以直接调用这些字符串处理函数来提高编程效率,要注意在使用字符串处理函数时,要在程序的开始处加上"#include <string.h>"头文件。

习　　题

一、选择题

1. 说明语句 int a[10]包括了_____个数组元素。

A) 10　　B) 11　　C) 1　　D) 不确定

2. 若有以下说明,则数值为 3 的表达式是_____ 。

```
int a[12]={1,2,3,4,5,6,7,8,9,10,11,12};
char c='a',e='b',g='c';
```

A) a[c-'a']　　B) a[3]　　C) a['e'-g]　　D) a['e'-c]

3. 定义如下变量和数组,则下面语句的输出结果是_____。

```
int i,x[3][3]={1,2,3,4,5,6,7,8,9};
for(i=0;i<3;i++) printf("%d",x[i][2-i]);
```

A) 1 5 9　　B) 1 4 7　　C) 3 5 7　　D) 3 6 9

4. 下列 4 种数组定义,合法的数组定义是_____。

A) char a[6]="string"　　B) int a[5]={0,1,2,3,4,5}

C) char a="string"　　D) int a[]={0,1,2,3,4,5}

5. 若有定义和语句:

```
char s[10];s="abcd";printf("%s\n",s);
```

则输出结果是_____。

A) abcd　　B) a　　C) ab cd　　D) 编译不通过

6. 若有以下程序为:

```
char a[]="xy\102\n\\\12";
printf("%d",strlen(str));
```

该程序的输出结果是_____。

A) 5　　B) 6　　C) 7　　D) 8

7. 有两个字符数组 a,b,则以下正确的输入语句是_____。

A) gets(a,b);　　B) scanf("%s%s",a,b);

C) scanf("%s%s",&a,&b);　　D) gets("a"),gets("b");

8. 下面程序段执行后,s 的值是_____。

```
char ch[]="600";
int i,s=0;
for(i=0;ch[i]>='0'&&ch[i]<='9';i++)
  s=10*s+ch[i]-'0';
```

A) 600　　B) 6　　C) 0　　D) 出错

9. 若有说明:int a[ ][3]={1,2,3,4,5,6,7,8,9,10};则 a 数组第一维的大小是_____。

A) 2　　B) 3　　C) 4　　D) 无确定值

10. 有下面程序段

```
char a[3],b[ ]="hello";
a=b;
printf("%s",a);
```

则将输出_____。

A) hello　　B) hel　　C) he　　D) 编译出错

11. 关于一维数组a的正确说明是_____。

A) int a(10);

B) int n=10,a[n];

C) int n;
scanf("%d",&n);
int a[n];

D) #define size 10
int a[size];

12. 若有说明:int x[3][4];,则对a数组元素的正确引用是_____。

A) x[2][4]　　B) x[1,3]　　C) x[4-2][0]　　D) x[0][4]

13. 判断字符串s1是否大于字符串s2,应当使用_____。

A) if(s1>s2)

B) if(strcmp(s1,s2))

C) if(strcmp(s2,s1)>0)

D) if(strcmp(s1,s2)>0)

二、填空题

1. 定义一个名为a的单精度实型一维数组,长度为4,所有元素的初值均为0的数据定义语句是_____。

2. 设有数据定义语句:int i=3, x[]={1,2,3,4};则数组x的数据类型是_____;该数组的长度是_____;数组元素x[i]的值是_____。

3. 有定义语句:char s[]="12345\0";则数组s的长度是_____;数组元素s[6]中存放的字符是_____。

4. 输入一组实数给一个3×3的矩阵,然后求出此矩阵两条对角线元素值的和,试将程序补充完整。

```
#include <stdio.h>
void  main(void)
{
    double a[3][3],sum=0;
    int i,j;
    for(i=0;i<3;i++)
      for(j=0;j<3;j++)
        scanf("%f", (1) );                    (1)_____
    for(i=0;i<3;i++)
      sum=sum+ (2) ;                          (2)_____
    printf("%f\n",sum);
}
```

5. 下列程序是统计某一个字符串中某个字符出现的次数,试将程序补充完整。

```
#include <stdio.h>
void main(void)
```

```
{ char str[100],c;
  int i,count;
  printf("输入一个字符串:");
  gets(str);
  printf("输入一个字符:");
  c=getchar();
     (1)     ;                                    (1)_____
  for(i=0;str[i]!='\0';i++)
    if(str[i]== c)
     (2)     ;                                    (2)_____
  printf("计数值=%d",count);
}
```

三、应用题

1. 已知下列程序,写出执行结果。

```
#include <stdio.h>
void main(void)
{
  int a[6],i;
  for(i=1;i<6;i++)
  { a[i]=9*(i-2+4*(i>5))%5;
    printf("%3d",a[i]);
  }
}
```

2. 阅读下列程序,写出执行结果。

```
#include <stdio.h>
void main(void)
{ int a[4]={5,16,7,14};
  int i;
  for(i=0;i<4;i++)
    a[i]+=i;
  for(i=3;i>=0;i--)
    printf("%3d",a[i]);
}
```

3. 阅读下列程序,写出执行结果。

```
#include <stdio.h>
void main(void)
{
  int i,j,a[4][4];
  for(i=0;i<4;i++)
```

```
    for(j=0;j<4;j++)   a[i][j]=i-j;
for(i=0;i<4;i++)
{
   for(j=0;j<4;j++)
     if(a[i][j]>0)   printf("%3d",a[i][j]);
   printf("\n");
}
```

四、编写程序题

1. 输入 10 个学生的分数，并计算平均分，打印低于平均分数的学生分数。

2. 统计从键盘输入的字符中每个数字、字母、空格及其他字符的个数。

3. 任意输入 20 个整数，统计其正数及负数的个数，计算并输出正数及负数之和。

4. 一个含有 10 个整数元素的数组，将这个数组中的元素按逆序重新存放后输出。

5. 输入矩阵 A(5 行 5 列)，完成下列要求：

输出矩阵 A，并显示行号和列号；将第 2 行和第 4 行元素对调后，输出新的矩阵 A。

6. 打印下面杨辉三角形。

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5   10   10    5    1
1    6   15   20   15    6    1
```

第7章　指　　针

【教学提示】

指针是C语言中广泛使用的一种数据类型。使用指针编程是C语言最主要的风格之一。利用指针变量可以有效地表示各种复杂的数据结构;能很方便地使用数组和字符串;并能直接处理内存地址,从而编出精练而高效的程序。本章要求掌握指针的概念、指针变量的定义和引用方法,掌握一维数组、二维数组和字符串指针等应用。

【核心概念】

指针变量　指针与数组　指针数组　指向指针的指针

7.1　指针的基本概念

7.1.1　内存地址和指针

在计算机中,我们所操作的数据都是存放在内存中的。内存是以字节为单位的一片连续的存储空间,为了便于管理,系统为每个字节编了一个号码,这个编号就叫做内存地址。

我们知道,不同的数据类型所占用的内存字节数不等。如整型量TC环境占2个字节(VC环境占4个字节),字符型量占1个字节等,在前面已有详细的介绍。为了正确地访问这些数据,也必须为它们编上号。C语言系统规定,某类型变量所占用的内存单元中第一个字节地址就是这个变量的地址。

根据前面的讨论我们知道,每个变量都有4个重要属性:

(1)名称;

(2)数据类型;

(3)值;

(4)地址。

例如:int　a=10;

这里我们定义了一个变量,系统为其分配2字节的内存单元。假设系统为其分配2字节的编号分别为1000H和1001H(地址用16进制表示),则变量的名称是a,数据类型是int,当前值是10,地址是1000H,如图7-1(a)所示。

有了地址,我们要访问某个变量就方便多了,除了用变量名直接访问外还可以通过地址间接访问。

(1)直接访问:要访问a的值,根据变量名a由系统找到内存单元1000H,获得其值10。

(2)间接访问:将a的地址1000H当作一个值存放在另一个内存单元内,假设其地址为2000H。要访问a的值,可先找到内存单元2000H,取出其值1000H,再根据该值找到相应内存单元,获得其值10,如图7-1(b)所示。

这就像到旅馆找某人,一般有两种情况:其一是知其房间,直接到房间去找他,这就是“直接访问”;其二是不知其房间,查登记簿获得房间号,再到房间去找他,这就是“间接访

问”。

这里我们可看到内存单元2000H存放的是一个地址，该地址指向变量a，我们可以通过地址能找到所指向的变量单元。因此在C语言中，把地址形象化地称为“指针”。

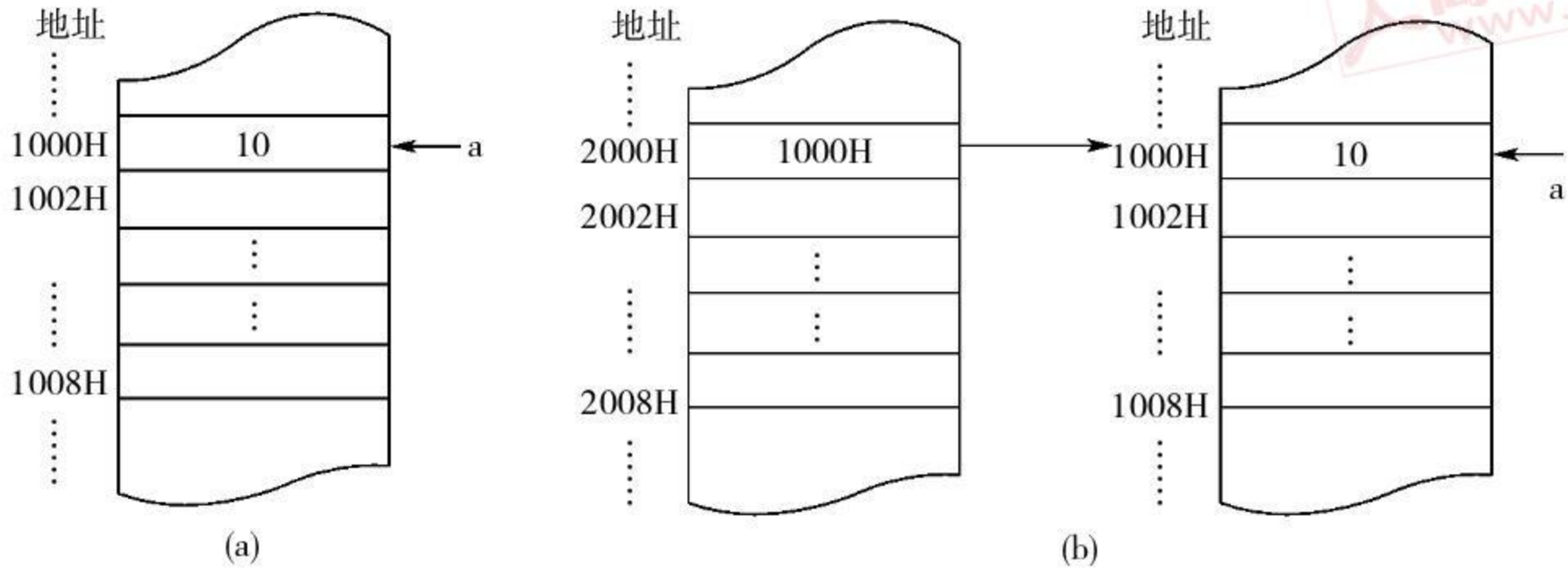


图7-1 内存单元与地址

指针是内存单元的地址，也是一种数据类型。因此，我们可以专门定义一个变量，此变量用来存放另一个变量的地址(即指针)，则称该变量为指针变量。

7.1.2 指针变量的定义

变量的指针就是变量的地址。存放变量地址的变量是指针变量，即在C语言中，允许用一个变量来存放指针，这种变量称为指针变量。因此，一个指针变量的值就是某个变量的地址或称为某变量的指针。

1. 定义

和普通变量一样指针变量也有名称、类型、地址和值。对指针变量的定义包括三个内容：

(1) 指针类型说明，即定义变量为一个指针变量。

(2) 指针变量名。

(3) 指针变量所指向变量的数据类型，也叫基类型。

指针变量说明的一般形式为：

类型说明符 *指针变量名；

其中：*是指针类型说明符，即指针标志，例如：

```
int *p1;
```

表示p1是一个指针变量，它的值是某个整型变量的地址。或者说p1指向一个整型变量。至于p1究竟指向哪一个整型变量，应由向p1赋予的地址来决定。又如：

```
double *p2;          /*p2是指向双精度浮点型变量的指针变量*/
float *p3;           /*p3是指向单精度浮点型变量的指针变量*/
char *p4;            /*p4是指向字符型变量的指针变量*/
```

说明：一个指针变量只能指向同基类型的变量，如p3只能指向单精度浮点型变量，不能指向一个双精度浮点型变量，或指向一个字符型变量等。

2. 指针的意义

在C语言中，一种数据类型或数据结构往往都占用一组连续的内存单元。如数组这种数据结构就是按元素顺序号连续存放的。用“地址”这个概念并不能很好地描述一种数据类

型或数据结构，而“指针”虽然实际上也只是一个地址，但它却可以是一个数据结构的首地址，而“指向”一个数据结构，因而概念更为清楚，表示更为明确。这也是引入“指针”概念的一个重要原因。

7.1.3　指针的运算

指针变量可以进行某些运算，但其运算的种类是有限的。它只能进行赋值运算和部分算术运算及关系运算。主要有：

(1)赋值运算；

(2)指针变量的取内容运算(间接访问运算)；

(3)与整数进行加减的算术运算；

(4)两个指针变量的相减运算；

(5)两个指针变量的关系运算。

1. 与指针相关的运算符

(1) 取地址运算符 &：取地址运算符 & 是单目运算符，其结合性为自右至左，优先级较高仅次于括号(见附录1)，其功能是取变量的地址。

(2) 取内容运算符 *：取内容运算符 * 是单目运算符，其结合性、优先级均与取地址运算符 & 相同，其功能用于取地址指向对象的数据。在 * 运算符之后跟的变量必须是指针变量。

【例7.1】 通过指针访问简单变量。

```
#include <stdio.h>
void main(void)
{ int a,b;                              /*定义了两个整型变量 a 和 b*/
  int *p1,*p2;                          /*定义了两个指针变量 p1 和 p2*/
  a=-10; b=10;                          /*给变量 a、b 分别赋值-10 和 10*/
  p1=&a; p2=&b;                         /*给变量 p1、p2 分别赋值 a 和 b 的地址*/
  printf("%d  %d\n",a,b);               /*分别输出 a 和 b 的值*/
  printf("%d  %d\n",*p1, *p2);          /*分别输出 p1 和 p2 所指向变量的值*/
}
```

程序运行结果，如图7-2所示。

图7-2　例7.1程序运行结果

【解析】

① 在程序第4行处虽然定义了两个指向整型变量的指针变量 p1 和 p2，但指针变量定义后一定要有指向，即存放某个变量的地址，故程序第6行的作用就是使 p1 指向 a，p2 指向 b，如图7-3所示。

② 最后一行的 * p1 就是变量 a，* p2 就是变量 b。最后两个 printf 函数作用是相同的。

③ 程序中有两处出现 * p1 和 * p2，请区分它们的不同含义。

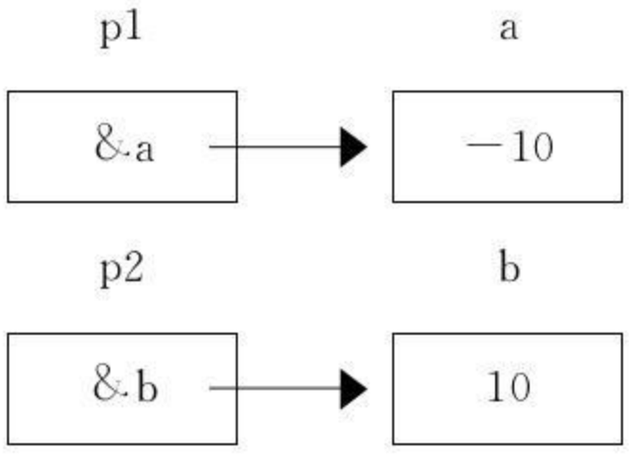


图 7-3　指针变量与变量的关系

第 4 行的“ * ”为指针类型说明标志符，第 8 行的“ * ”为指针指向运算符。需要注意的是它们都不是乘法运算符。在指针变量说明中出现的“ * ”是说明标志符，表示其后的变量是指针类型。而表达式中出现的“ * ”则是一个运算符，用以表示指针变量所指的变量。

根据运算符 * 和 & 的规定，我们知道 * (&a)的含义是先用地址运算符 & 求出 a 的地址，然后用 * 间接访问运算符取出该地址中存放的内容，即 a 的值。也可写作 * &a。

2. 赋值运算

指针变量同普通变量一样，使用之前不仅要定义说明，而且必须赋予具体的值。给指针变量赋值的方式也是多种多样的，但由于指针变量的特殊性，赋值的方式却不完全与普通变量相同。下面将给予说明。

(1)赋地址值：把一个变量的地址赋予指针变量。

【例 7.2】　实例。

```
#include <stdio.h>
void main(void)
{ int   i=10, *p;
  p=&i;
  printf("%d", *p);
}
```

图 7-4　例 7.2 程序运行结果

程序中的第 3 行将整型变量 i 的地址赋予了 p，运行结果如图 7-4 所示。

(2)初始化赋值：例如：

```
#include <stdio.h>
void main(void)
{ int   i=10, *p=&i;
  printf ("%d", *p);
}
```

第 2 行在定义指针变量 p 的同时赋 i 的地址。

(3)赋指针变量的值：把一个指针变量的值赋予有相同基类型的另一个指针变量。例如：

```
int   a=10, *pa=&a, *pb;
pb=pa;          /* 把 pa 的值赋予指针变量 pb */
```

由于pa、pb均为指向整型变量的指针变量，因此可以相互赋值。由于pa已指向变量a，所以经过pb＝pa；后，pb获得的pa值也指向变量a了，如图7－5所示。

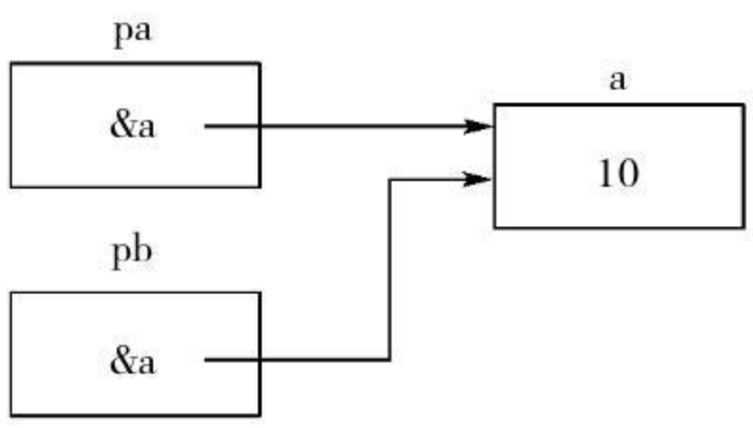


图7－5　指针变量相互赋值

(4) 赋数组的首地址：把数组的首地址赋予与数组有相同基类型的指针变量，如图7－6所示。例如：

```
int   a[5], * pa;
pa＝a;                    / * C规定，数组名为数组的首地址。但pa＝&a;是错误的 * /
```

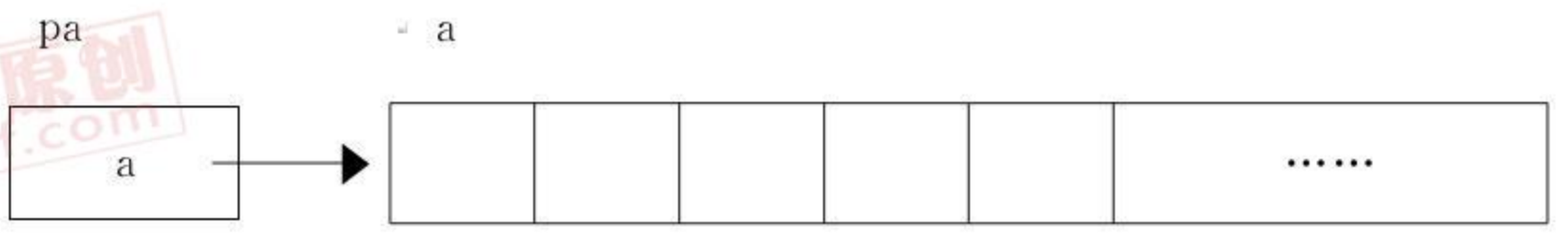


图7－6　数组的首地址赋给指针

当然也可采取初始化赋值的方法：

```
int   a[5], * pa＝a;
```

在下一节还将详细介绍。

(5)赋字符串首地址：把字符串的首地址赋予指向字符类型的指针变量。例如：

```
char    * pc;
pc＝"I am a student.";
```

或用初始化赋值的方法写为：

```
char    * pc＝"I am a student.";
```

这里要注意的是并不是把整个字符串装入指针变量，而是把存放该字符串的字符数组的首地址装入指针变量。

(6)动态分配内存单元：malloc函数可在空闲内存中开辟一块指定大小且连续的内存空间，并将此空间的起始地址作为函数值返回。其一般形式：

```
(type_name * )malloc(Byte_size)
```

type_name是存放数据的类型，Byte_size是希望分配的内存单元的大小。例如：

```
p＝(int * )malloc(2);
```

语句通过malloc函数在空闲内存中开辟一块2个字节的连续空间，并将此空间的起始地址赋予指针变量p，如图7－7所示。

(7) 赋空值：空指针是由对指针变量赋予0值而得到的。例如：

```
#define NULL   0
int * p＝NULL;    / * 事实上NULL是符号常量，其值就为整数0，所以也可以不用上一行 * /
```

对指针变量赋0值和不赋值是不同的。指针变量未赋值时，其值是任意的，是不能使用

的,否则可能造成意外错误。而指针变量赋0值后,则可以使用,只是它不指向具体的变量而已,如图7-8所示。

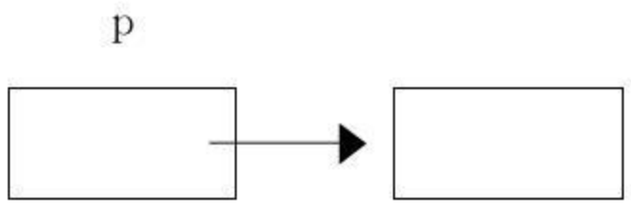


图7-7 malloc函数的动态分配

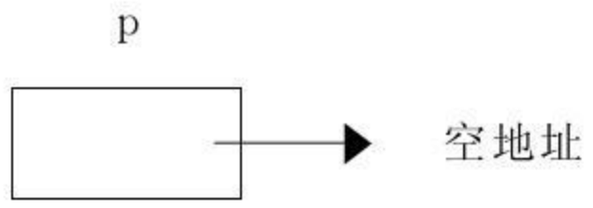


图7-8 赋空值

说明:

(1)每个指针变量只能赋一固定类型的变量的地址,即只能指向一固定类型的变量的内存单元。

(2)指针变量赋空值与不赋值有本质的区别。未经赋值的指针变量不能使用,否则将造成系统混乱,甚至死机。

(3)指针变量虽为整数值,但不能直接赋整数值。在C语言中,变量的地址是由编译系统分配的,对用户完全透明,用户不知道变量的具体地址。

3. 与整数进行加减的算术运算

指针变量加或减一个整数n的意义是把指针指向的当前位置向前或向后移动n个位置。例如:

```
int  j, * p=&j;
p=p+3;          /* 指针 p 加 3 */
```

指针在内存中移动3个单位长度(即指针在内存中移动到后面的第三个整数,也即指针中存储的地址值加3个单位长度),每个单位长度不一定是1个字节,而是指针指向的数据类型在内存中占用的字节个数,如图7-9所示。

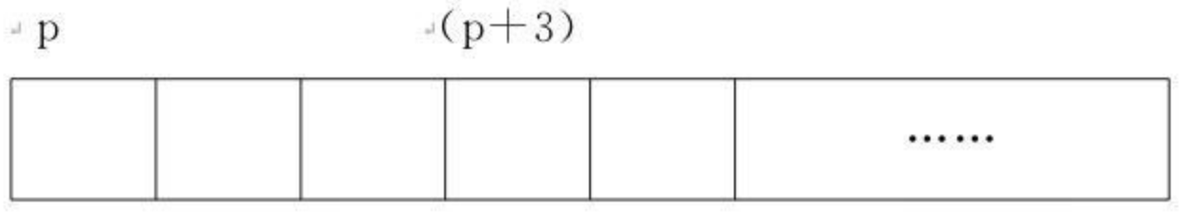


图7-9 指针p加3

例如,sizeof(int)为2。

说明:sizeof取某类型变量所占字节数运算符,格式为sizeof(变量的类型名),则p+3的值为p的值加2*3个字节数。

作为特例,指针变量可进行自加、自减运算。

例如,p++;p--;

说明:

(1)只能加减整数;

(2)只有当指针指向一串连续的存储单元时,指针移动才有意义;

(3)在对指针进行加减时不是简单地加减一个数字,而是加减若干个存储单元的长度,(其中1个存储单元的长度占多少字节,视指针的基类型由系统确定)。

4. 两个指针变量的相减运算

可以对指向同一串连续存储单元的两个指针进行相减运算,两指针变量相减所得之差是两个指针所指两数据之间相差的数据个数。实际上是两个指针值(地址)相减之差再除以

该数据的长度(字节数)。例如:

```
float  a[10], * pf1, * pf2;
pf1=&a[0],pf2=&a[4];
```

假设a的第0个元素地址为2000,那么pf1的值是2000,pf2的值是2016,而浮点数组每个元素占4个字节,于是pf2-pf1的结果为(2016-2000)/4=4,即表示pf2和pf1之间相差4个元素,如图7-10所示。

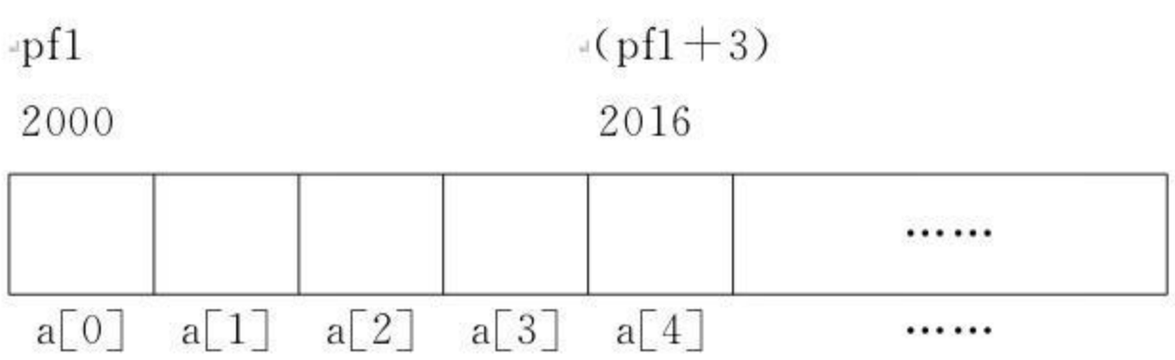


图7-10　pf1与pf2之间的关系

说明:

(1)两个指针变量不能进行加法运算。

(2)多个指针指向同一个目标或一串连续的存储单元时,相减才有意义。

(3)相减主要是查看两数组元素之间的距离。

5. 两个指针变量的关系运算

两个指针变量可以比较大小,从而查看地址的高低。但只有在多个指针指向同一个目标或一串连续的存储单元时,比较才有意义。

假如pf1和pf2指向同一数组的某元素,则:

```
pf1==pf2/ * 表示pf1和pf2指向同一数组元素 * /
pf1>pf2 * /表示pf1处于高地址位置 * /
pf1<pf2 * /表示pf1处于低地址位置 * /
```

指针变量还可以与0比较。

设p为指针变量,则p==0表明p是空指针,它不指向任何变量;

总结:试分别说明下面各行的含义。

```
int  i=5,s,t, * p, * q;
int  a,b, * pa=&a, * pb=&b;
p=&i;                    /*  给指针变量p赋值,使p指向变量i */
q=p;                     /*  给指针变量q赋值,使q与p指向同一个变量i */
 * p= * p+1;             /*  等价于( * p)++, * p+=1 */
p++; p--;                /*  指针移动 */
 * p++;                  /*  假设有意义,等价于 * (p++) */
s= * pa+ * pb;           /*  求a+b之和,( * pa就是a, * pb就是b) */
t= * pa * * pb;          /*  求a * b之积 */
 * &i=10;                /*  * &i等价于 * (&i)即i */
&* p=&a; * &p=&a;/*  &* p与 * &p都等于p */
```

7.2 指针与数组

一个变量有一个地址，一个数组包含若干元素，每个数组元素(下标变量)都在内存中占用存储单元，它们都有相应的地址，且这些数组元素占用一块连续的内存空间，也有相应的地址。我们把数组的起始地址叫做数组的指针，数组元素的地址叫做数组元素的指针。

7.2.1 指针与一维数组

一个数组是由各个数组元素(下标变量)组成的，占用一块连续的内存单元。每个数组元素按其类型不同占用几个连续的内存字节，其地址是指它所占内存第一个字节的地址。

1. 指向一维数组元素的指针

定义一个指向一维数组元素的指针变量的方法，与以前介绍的指针变量相同。例如：

```
int  a[10];      /* 定义 a 为包含 10 个整型数据的数组 */
int  *p;         /* 定义 p 为指向整型变量的指针 */
```

应当注意，因为数组为 int 型，所以指针变量也应为指向 int 型的指针变量。下面是对指针变量赋值：

```
p=&a[0];
```

把 a[0]元素的地址赋给指针变量 p。也就是说，p 指向 a 数组的第 0 号元素，如图 7-11 所示。

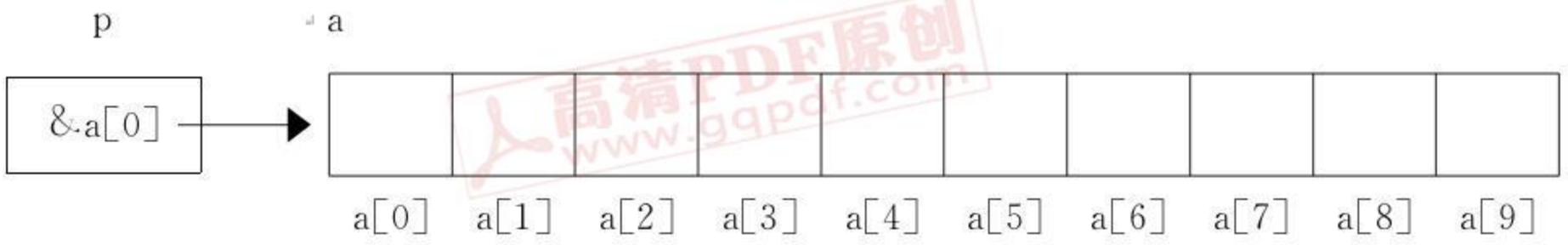


图 7-11 将数组的首地址赋给指针

C 语言规定，数组名代表数组的首地址，也就是第 0 号元素的地址。因此，p=&a[0]；等价于下面的语句：

```
p=a;
```

在定义指针变量时可以赋给初值：

```
int  a[10],*p=&a[0];
```

它等价于下面两行：

```
int  a[10],*p;
p=&a[0];
```

当然定义时也可以写成：

```
int  a[10],*p=a;
```

从图 7-11 中我们可以看出有以下关系：

p,a,&a[0]均指向同一单元，它们是数组 a 的首地址，也是 0 号元素 a[0]的地址。应该说明的是，p 是变量能赋值，而 a,&a[0]都是常量不能赋值。在编程时应特别予以注意。

2. 通过指针引用数组元素

由上一节的讨论我们知道，如果指针变量 p 已指向数组中的一个元素，则 p+1 指向同

一数组中的下一个元素。

引入指向数组的指针变量后，就可以用两种方法来访问数组元素了。

例如：int　a[10]，*p=&a[0];/*或p=a;*/

(1) p和a就是a[0]的地址，p+1和a+1就是a[1]的地址，…依此类推p+i和a+i就是a[i]的地址了，或者说它们指向a数组的第i个元素，如图7-12所示。

(2) *(p+i)或*(a+i)就是p+i或a+i所指向的数组元素，即a[i]。例如，*(p+9)或*(a+9)就是a[9]。

(3) C语言规定，指向数组的指针变量也可以带下标，如p[i]与*(p+i)等价。

根据以上叙述，访问一维数组的一个元素a[i]还可以用：*(a+i)通过数组的首地址访问；*(p+i)通过指针变量来访问；p[i]用带下标的指针变量访问。即可以用下标法和指针法。

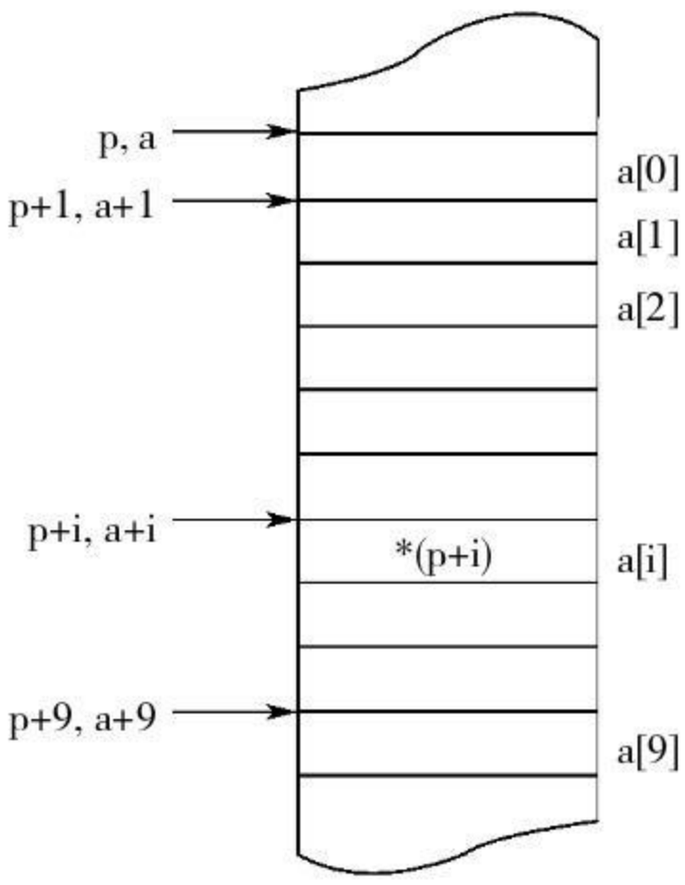


图7-12　地址的表示

下标法：即用a[i]或p[i]形式访问数组元素。

指针法：即采用*(a+i)或*(p+i)形式，用间接访问的方法来访问数组元素。

【例7.3】 设有一个具有10个元素的整型数组，输出数组中的全部元素。

(1)方法一(通过数组元素)

```
#include <stdio.h>
void main(void)
{ int a[10],i;
  for(i=0;i<10;i++)
    a[i]=2*i;
  for(i=0;i<10;i++)
    printf("a[%d]=%d\n",i,a[i]);
}
```

(2)方法二(通过指针变量名)

```
#include <stdio.h>
void main(void)
{ int a[10],i,*p=a;
  for(i=0;i<10;i++)
    p[i]=2*i;
  for(i=0;i<10;i++)
    printf("a[%d]=%d\n",i,p[i]);
}
```

(3)方法三(通过数组名)

```
#include <stdio.h>
void main(void)
{ int a[10],i;
```

```
    for(i=0;i<10;i++)
       *(a+i)=2*i;
    for(i=0;i<10;i++)
      printf("a[%d]=%d\n",i,*(a+i));
}
```

(4)方法四(通过指针变量)

```
#include <stdio.h>
void main(void)
{ int a[10],i,*p=a;
  for(i=0;i<10;i++)
     *(p+i)=2*i;
  for(i=0;i<10;i++)
    printf("a[%d]=%d\n",i,*(p+i));
}
```

(5)方法五(较精练高效的程序)

```
#include <stdio.h>
void main(void)
{ int a[10], i=0, *p=a;
  while(i<10)
  {  *p=2*i;
    printf("a[%d]=%d\n",i++,*p++);
  }
}
```

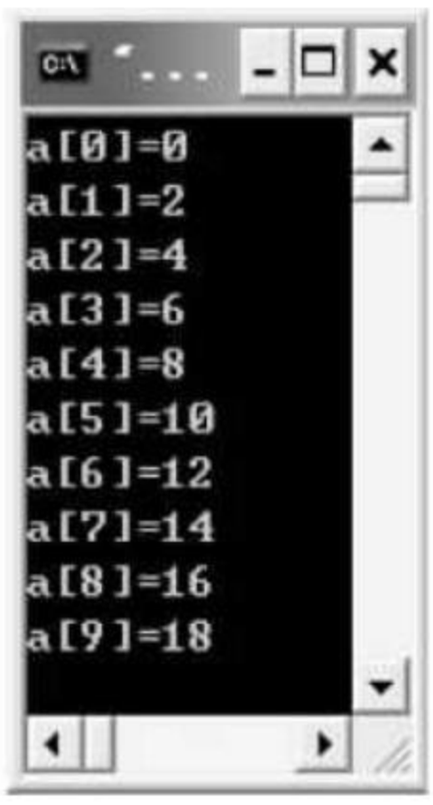


图 7-13　例 7.3 程序运行结果

五种方法程序的运行结果,如图 7-13 所示。

上面五个程序执行的结果是完全一样的,请比较它们的不同点,进一步认识访问数组元素的不同方法。

几个注意的问题:

(1)指针变量可以实现本身的值的改变。如 p++是合法的;而 a++是错误的。因为 a 是数组名,它是数组的首地址,是常量。

(2)从上例可以看出,虽然定义数组时指定它包含 10 个元素,但指针变量可以指到数组以后的内存单元,系统并不认为非法。例如方法五,当循环结束后 p 已指向数组以后 a 的内存单元。

(3)要注意每次循环时指针变量的当前值。

(4)要注意指针变量运算的优先级和结合性。

① *p++,由于++和*优先级相同,结合方向也相同自右而左,故其等价于*(p++);而(*p)++表示 p 所指向的元素值加 1。

② *(p++)与*(++p)作用不同。若 p 的初值为 a,则*(p++)等价 a[0],*(++p)等价 a[1]。

③ 如果 p 当前指向 a 数组中的第 i 个元素,则

*(p－－)等价于 a[i－－];

*(＋＋p)等价于 a[＋＋i];

*(－－p)等价于 a[－－i]。

7.2.2　指针与二维数组

指针变量可以指向一维数组中的元素,同样也可以指向多维数组中的元素。但多维数组的指针要比一维数组的指针复杂些。本小节着重以二维数组为例介绍多维数组的指针变量。

1. 二维数组的地址

设有整型二维数组 a[3][4]的定义如下:

int　a[3][4]={0,1,2,3,10,11,12,13,20,21,22,23}

假设数组 a 的首地址为 1000,则各下标变量的首地址及其值如图 7－14 所示。

前面介绍过,C 语言允许把一个二维数组分解为多个一维数组来处理。因此数组 a 可分解为三个一维数组,即 a[0],a[1],a[2]。每一个一维数组又含有四个元素,如图 7－15 所示。

1000	1002	1004	1006
0	1	2	3
1008	1010	1012	1014
10	11	12	13
1016	1018	1020	1022
20	21	22	23

图 7－14　二维数组存放的地址

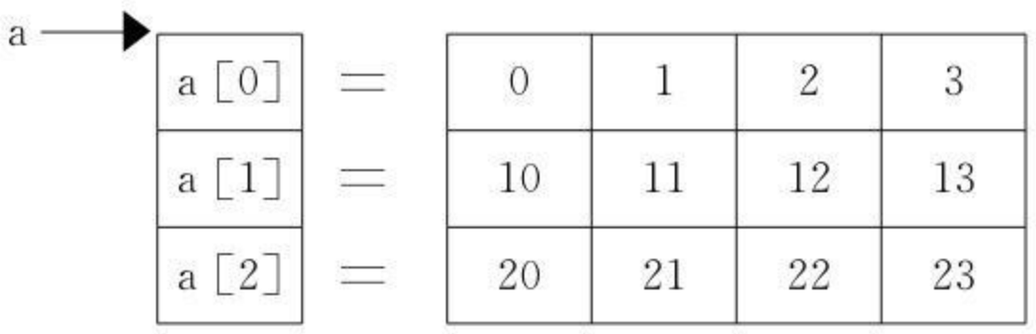


图 7－15　二维数组分解成三个一维数组

例如:a[0]数组,含有 a[0][0],a[0][1],a[0][2],a[0][3]四个元素。

数组及数组元素的地址表示如下:

从二维数组的角度来看,a 是二维数组名,a 代表整个数组的首地址,也是第 0 号一维数组 a[0]的首地址,即二维数组 0 行的首地址,等于 1000。a＋1 代表第 1 号一维数组 a[1]的首地址,即二维数组 1 行的首地址,等于 1008。依此类推,a＋2 代表 2 行的首地址,等于 1016,如图 7－16 所示。

图 7－16　每一行对应的首地址

a[0]是第一个一维数组的数组名和首地址,因此也为 1000。*(a＋0)或 *a 是与 a[0]等效的,它表示一维数组 a[0]的 0 号元素的首地址,也为 1000。&a[0][0]是二维数组 a 的 0 行 0 列元素首地址,同样是 1000。因此,a,a[0],*(a＋0),*a,&a[0][0]的值是相等的。但它们所指向的数据类型不同。a 指向含 2*4 个字节的数据(数组 a 一行,四个整型变量),a[0],*(a＋0),*a,&a[0][0]都指向含 2 个字节的数据(数组 a 一个元素,一个整型

变量)。特别要注意的是,*(a+0),*a是指针,而不是元素。

同理,a+1是二维数组1行的首地址,等于1008。a[1]是第二个一维数组的数组名和首地址,因此也为1008。&a[1][0]是二维数组a的1行0列元素地址,也是1008。因此a+1,a[1],*(a+1),&a[1][0]的值是相等的。

由此类推可得出:a+i,a[i],*(a+i),&a[i][0]的值是相等的。

此外,&a[i]和a[i]也是等同的。因为在二维数组中不能把&a[i]理解为元素a[i]的地址,不存在元素a[i]。C语言规定,它是一种地址计算方法,表示数组a第i行首地址。由此,我们得出:a[i],&a[i],*(a+i)和a+i的值也是相等的。

另外,a[0]也可以看成是a[0]+0,是一维数组a[0]的0号元素的地址,而a[0]+1则是a[0]的1号元素地址,由此可得出a[i]+j则是一维数组a[i]的j号元素地址,它等于&a[i][j],如图7-17所示。

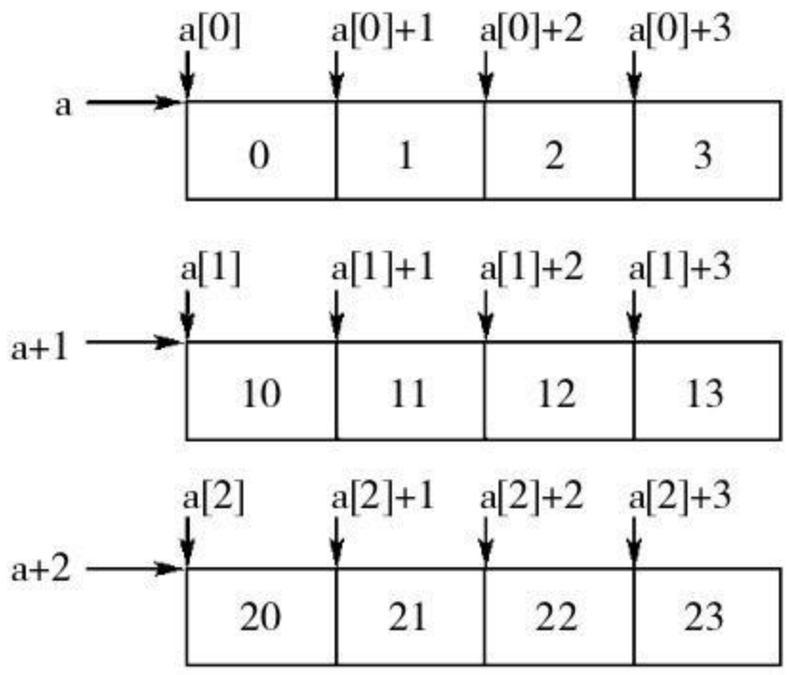


图7-17 用a[i]+j表示地址

由a[i]=*(a+i)得a[i]+j=*(a+i)+j。由于*(a+i)+j是二维数组a的i行j列元素的首地址,所以,该元素的值等于*(*(a+i)+j)。

二维数组中各种地址与元素的不同形式,见表7-1所列。

表7-1 二维数组中各种地址与元素

表示形式	含义
a	二维数组名,数组首地址;第0行的地址
a+i,&a[i]	第i行的地址
a[0],*(a+0),*a	第0行第0列元素地址
a[i]+j,*(a+i)+j,&a[i][j]	第i行第j列元素地址
(a[i]+j),(*(a+i)+j),a[i][j]	第i行第j列元素值

【例7.4】 二维数组中各种地址与元素输出实例。

```
#include <stdio.h>
void main(void)
{ int a[3][4]={0,1,2,3,10,11,12,13,20,21,22,23};
  printf("%x,",a);
  printf("%x,",xa);
```

```
    printf("%x,",a[0]);
    printf("%x\n",&a[0][0]);
    printf("%x,",a+1);
    printf("%x,",*(a+1));
    printf("%x,",&a[1]);
    printf("%x\n",&a[1][0]);
    printf("%x,",a[1]+1);
    printf("%x\n",*(a+1)+1);
    printf("%d,%d\n",*(a[1]+1),*(*(a+1)+1));
}
```

程序运行结果，如图 7－18 所示。

图 7－18　例 7.4 程序运行结果

2. 指向多维数组的指针变量

(1) 指向元素的指针变量

我们知道数组元素也是一个变量，而二维数组在内存中是按行连续存放的，因此可以用指针变量指向其中的元素，通过指针变量来访问二维数组的元素，如图 7－19 所示。

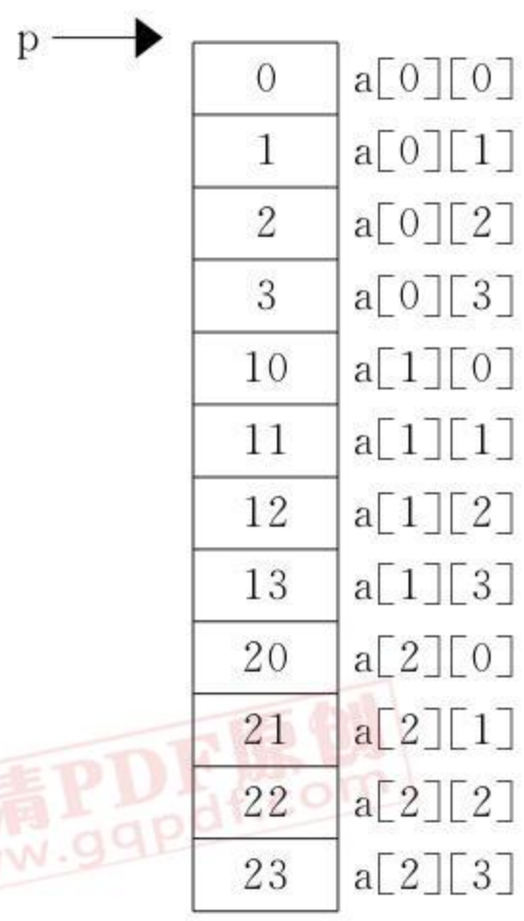


图 7－19　用指针变量表示数组元素

【例 7.5】　利用指向元素的指针变量输出数组的所有元素。

```
#inclnde <stdio.h>
void main(void)
{ int a[3][3]={ 0,1,2,10,11,12,20,21,22};
  int *p;
  for(p=a[0];p<=a[0]+8;p++)
  { if((p-a[0])%3==0)
      printf("\n");
    printf("%4d ",*p);
  }
}
```

图 7－20　例 7.5 程序运行结果

程序运行结果，如图 7－20 所示。

【解析】　p 是指向整型变量的指针变量，它可以指向一般的整型变量，也可以指向整型数组的元素。程序中通过第 4 行中的 p=a[0];使 p 指向数组的第一个元素，通过第 4 行中的 p++使指针下移一个元素。其中 p=a[0]也可写作 p=&a[0][0]。

上例利用循环输出全部元素。如果要输出i行j列的元素,可利用a[i][j]在数组中位置计算公式:

i * m+j

计算出该元素是数组的第i * m+j个元素,其中m是二维数组的列数(数组大小为n×m)。于是,若执行了p=a[0];则p+i * m+j就是a[i][j]的地址,*(p+i * m+j)就是元素a[i][j]。如a[1][2]就是第6(1×4+2=6,元素从第0个开始)个元素为12。

(2) 指向数组的指针变量

把二维数组a分解为一维数组a[0],a[1],a[2]之后,设p为指向二维数组中某个一维数组a[0],a[1],a[2]的指针变量。可定义为:

int (*p)[4];

它表示p是一个指针变量,它指向包含4个元素的一维数组。若有语句int a[3][4];p=a;则p指向第一个一维数组a[0],其值等于a、a[0]或&a[0][0]。而p+i则指向一维数组a[i],这里要注意a[i]不是二维数组的某元素,而是第i行,包含4个元素。由于p+i是指向二维数组行a[i]的,所以也把p叫行指针,如图7-21所示。

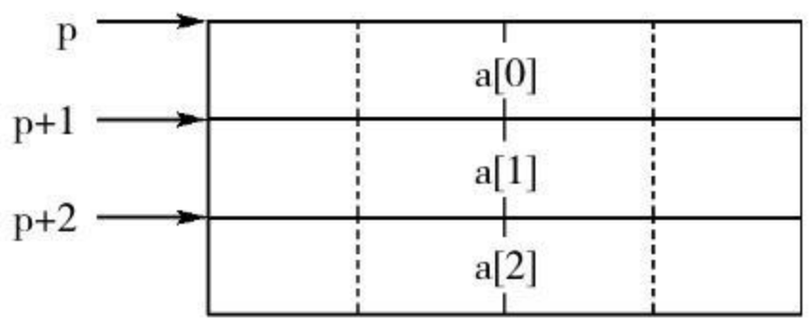


图7-21 p为行指针

从前面的分析可看出p与a的基类型是一致的。p+i是一维数组a[i]的首地址,而*(p+i)是一维数组a[i]第0个元素的地址,*(p+i)+j就是二维数组i行j列的元素的地址,而*(*(p+i)+j)则是i行j列元素的值。另外,C语言规定p[i][j]相当于a[i][j]。

二维数组行指针变量说明的一般形式为:

类型说明符(*指针变量名)[下标]

【例7.6】 利用行指针变量输出数组的所有元素。

```
#include <stdio.h>
void main(void)
{ int a[3][4]={ 0,1,2,3,10,11,12,13,20,21,22,23};
  int i,j,(*p)[4];
  p=a;
  for(i=0;i<3;i++)
  { for(j=0;j<4;j++)
      printf("%4d ",*(*(p+i)+j));
    printf("\n");
  }
}
```

程序运行结果,如图7-22所示。

图7-22 例7.6程序运行结果

二维数组中行指针名和数组名的表示形式，见表 7-2 所列。

表 7-2　二维数组中指针表示各种元素

行指针名表示形式	数组名表示形式	含　　义
p	a	第 0 行的地址
p+i	a+i,&a[i]	第 i 行的地址
p[0],*(p+0),*p	a[0],*(a+0),*a	第 0 行第 0 列元素地址
p[i]+j,*(p+i)+j	a[i]+j,*(a+i)+j,&a[i][j]	第 i 行第 j 列元素地址
(p[i]+j),(*(p+i)+j),p[i][j]	*(a[i]+j),*(*(a+i)+j),a[i][j]	第 i 行第 j 列元素值

7.2.3　指针与字符串

1. 字符串的表示形式

在 C 语言中，可以用两种方法访问一个字符串。

(1)用字符数组存放一个字符串。

【例 7.7】　输出字符串。

```
#include <stdio.h>
void main(void)
{ char s[]="I am a student.";
  printf("%s\n",s);
}
```

图 7-23　例 7.7 程序运行结果

程序运行结果，如图 7-23 所示。字符数组的存放形式如图 7-24 所示。

s

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]	s[11]	s[12]	s[13]	s[14]	s[15]
I		a	m		a		s	t	u	d	e	n	t	.	\0

图 7-24　字符数组的存放形式

【解析】　这里定义了一个数组 s[16]，元素的个数要大于字符串长度，至少是字符串长度加 1。第二行 char s[]="I am a student.";和前面介绍的数组属性一样，s 是数组名，它代表字符数组的首地址，数组的每个元素存放一个字符。

(2)用字符指针指向一个字符串。

【例 7.8】　输出字符串。

```
#include <stdio.h>
void main(void)
{ char    *st="I am a student.";
  printf("%s\n",st);
}
```

st

I		a	m		a		s	t	u	d	e	n	t	·	\0

图 7-25 字符串的存放形式

程序运行结果,如图 7-26 所示。

图 7-26 例 7.8 程序运行结果

这里定义了一个字符指针变量 st,赋初值使其指向字符串常量。第二行也可写成 char *st;st="I am a student.";C 语言规定,字符串常量就代表该字符串的首地址。但【例 7.7】第三行不能写成 char s[16];s="I am a student.";因数组名是地址常量,不能赋值。

同时要注意,字符指针变量 st 可指向其他字符。如:

```
char  ch, *pc=&ch;
```

这里字符指针变量 pc 指向字符变量 ch。

```
pc="I love China!  ";
```

则是字符指针变量 pc 指向一个字符串。把字符串的首地址赋予了 pc。

【例 7.9】 输出字符串中 n 个字符后的所有字符。

```
#include <stdio.h>
void main(void)
{ char *ps="this is a book";
  int n=10;
  ps=ps+n;
  printf("%s\n",ps);
}
```

图 7-27 例 7.9 程序运行结果

程序运行结果,如图 7-27 所示。

在程序中对 ps 初始化时,即把字符串首地址赋予 ps,当 ps= ps+10 之后,ps 指向字符“b”,因此输出为"book"。

2. 使用字符指针变量与字符数组的区别

用字符数组和字符指针变量都可实现字符串的存储和运算,但是两者是有区别的。在使用时应注意以下几个问题:

(1)字符指针变量本身是一个变量,只能用于存放字符变量的地址。用来处理字符串时,是用于存放字符串的首地址。而字符串本身是存放在以该首地址为首的一块连续的内存空间中,并以‘\0’作为串的结束标志。字符数组是有若干个数组元素组成的,它可直接用来存放整个字符串。

(2)对字符指针方式

```
char *ps="C Language";
```

可以写为:

```
char * ps;
ps="C Language";
```

而对数组方式：

```
char  st[]={"C Language"};
```

不能写为：

```
char  st[];
st={"C Language"};
```

而只能对字符数组的各元素逐个赋值。

(3)char * ps,st[20];字符指针变量名 ps 既可指向数组的开始,也可以指向数组的任一元素或数组外的某字符变量,字符数组名 st 永远是数组 st[20]的首地址,不能指向其他地方,是常量。

从以上几点可以看出字符串指针变量与字符数组在使用时的区别,同时也可看出使用指针变量更加方便。

7.3　指针数组和指向指针的指针

7.3.1　指针数组

一个数组的元素值均为指针,则称为指针数组。指针数组是一组有序的指针集合。指针数组的所有元素都必须是指向相同数据类型的指针变量。

指针数组说明的一般形式为：

类型说明符　* 指针变量名[下标]

例如:int * pa[3];

表示 pa 是一个指针数组,它有三个数组元素,每个元素的值都是一个指针,指向整型变量。

通常可用一个指针数组来指向一个二维数组。指针数组中的每个元素被赋予二维数组每一行的首地址,因此也可理解为指向一个一维数组。例如：

```
int a[3][4]={ 0,1,2,3,10,11,12,13,20,21,22,23};
int * pa[3]={a[0],a[1],a[2]};
```

指针数组 pa 中的每个元素指向二维数组 a 的某行第 0 个元素,如图 7-28 所示。

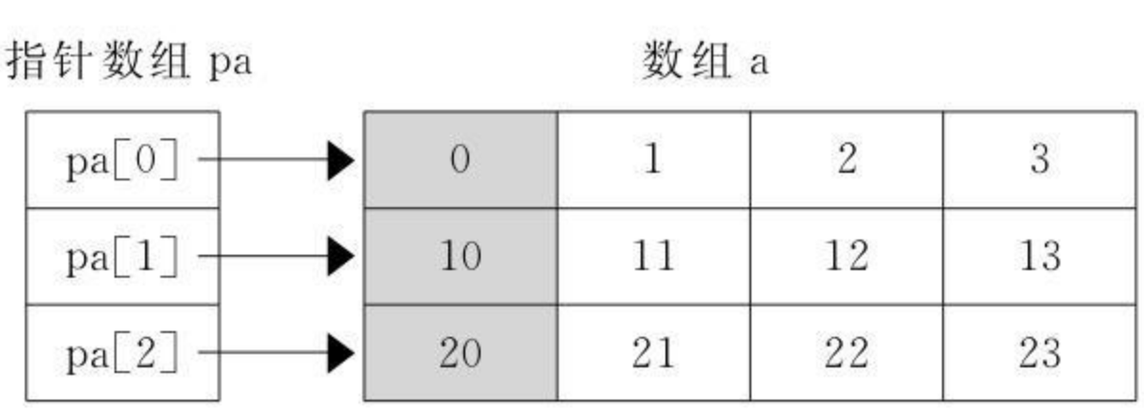


图 7-28　指针数组

【例 7.10】　实例。

```
#include <stdio.h>
void main(void)
```

```
{ int a[3][3]={1,2,3,4,5,6,7,8,9};
  int i ,*pa[3],*p=a[0];
  for(i=0;i<3;i++)
    pa[i]=a[i];
  for(i=0;i<3;i++)
    printf("%d,%d,%d\n",a[i][2-i],*a[i],*(*(a+i)+i));
  for(i=0;i<3;i++)
    printf("%d,%d,%d\n",*pa[i],p[i],*(p+i));
}
```

程序运行结果,如图 7-29 所示。

图 7-29 例 7.10 程序运行结果

【解析】 本例程序中,pa 是一个指针数组,用循环语句 for(i=0;i<3;i++) pa[i]=a[i];使三个元素分别指向二维数组 a 的各行,然后用循环语句输出指定的数组元素。其中 *a[i]表示 i 行 0 列元素值;*(*(a+i)+i)表示 i 行 i 列的元素值;*pa[i]表示 i 行 0 列元素值;由于 p 与 a[0]相同,故 p[i]表示 0 行 i 列的值;*(p+i)表示 0 行 i 列的值。读者可仔细领会元素值的各种不同的表示方法。

应该注意指针数组和二维数组指针变量的区别。两者虽然都可用来指向二维数组,但是其表示方法和意义是不同的。

二维数组指针变量是单个的变量,其一般形式中"(*指针变量名)"两边的括号不可少。而指针数组类型表示的是多个指针(一组有序指针)在一般形式中"*指针数组名"两边不能有括号。例如:

int (*p)[3];

表示一个指向二维数组的指针变量。该二维数组的列数为 3 或分解为一维数组的长度为 3。

int *p[3];

表示 p 是一个指针数组,有三个下标变量 p[0],p[1],p[2]均为指针变量。

指针数组也常用来表示一组字符串,这时指针数组的每个元素被赋予一个字符串的首地址。指向字符串的指针数组的初始化更为简单。

若有 int a[N][M],*pr[N]={a[0],a[1],a[2],…,a[N-1]};(其中 N、M 是常整数)则二维数组中指针表示各种地址,见表 7-3 所列。

表 7-3 二维数组中指针表示各种地址

指针数组名表示形式	数组名表示形式	含 义
pr+i	a[i],*(a+i)	第 i 行第 0 列元素地址
pr[0],*(pr+0),*pr	a[0],*(a+0),*a	第 0 行第 0 列元素地址
pr[i]+j,*(pr+i)+j	a[i]+j,*(a+i)+j,&a[i][j]	第 i 行第 j 列元素地址
(pr[i]+j),(*(pr+i)+j),pr[i][j]	*(a[i]+j),*(*(a+i)+j),a[i][j]	第 i 行第 j 列元素值

【例 7.11】 用指针数组来指向一组字符串,当输入一个 1～7 之间的整数,输出对应的

星期名。

```
#include <stdio.h>
void main(void)
{ int i;
  char *week_name[]={ "Illegal day","Monday","Tuesday",
                      "Wednesday","Thursday","Friday",
                      "Saturday","Sunday"
                    };
  printf("input Day No:\n");
  scanf("%d",&i);
  if (i<1||i>7)
    printf("Day No:%2d-->%s\n",i, week_name[0]);
  else
    printf("Day No:%2d-->%s\n",i, week_name[i]);
}
```

第4行至第6行完成这个初始化赋值之后，week_name[0]即指向字符串"Illegal day"，week_name[1]指向"Monday"……，各字符串的存放形式，如图7-30所示。

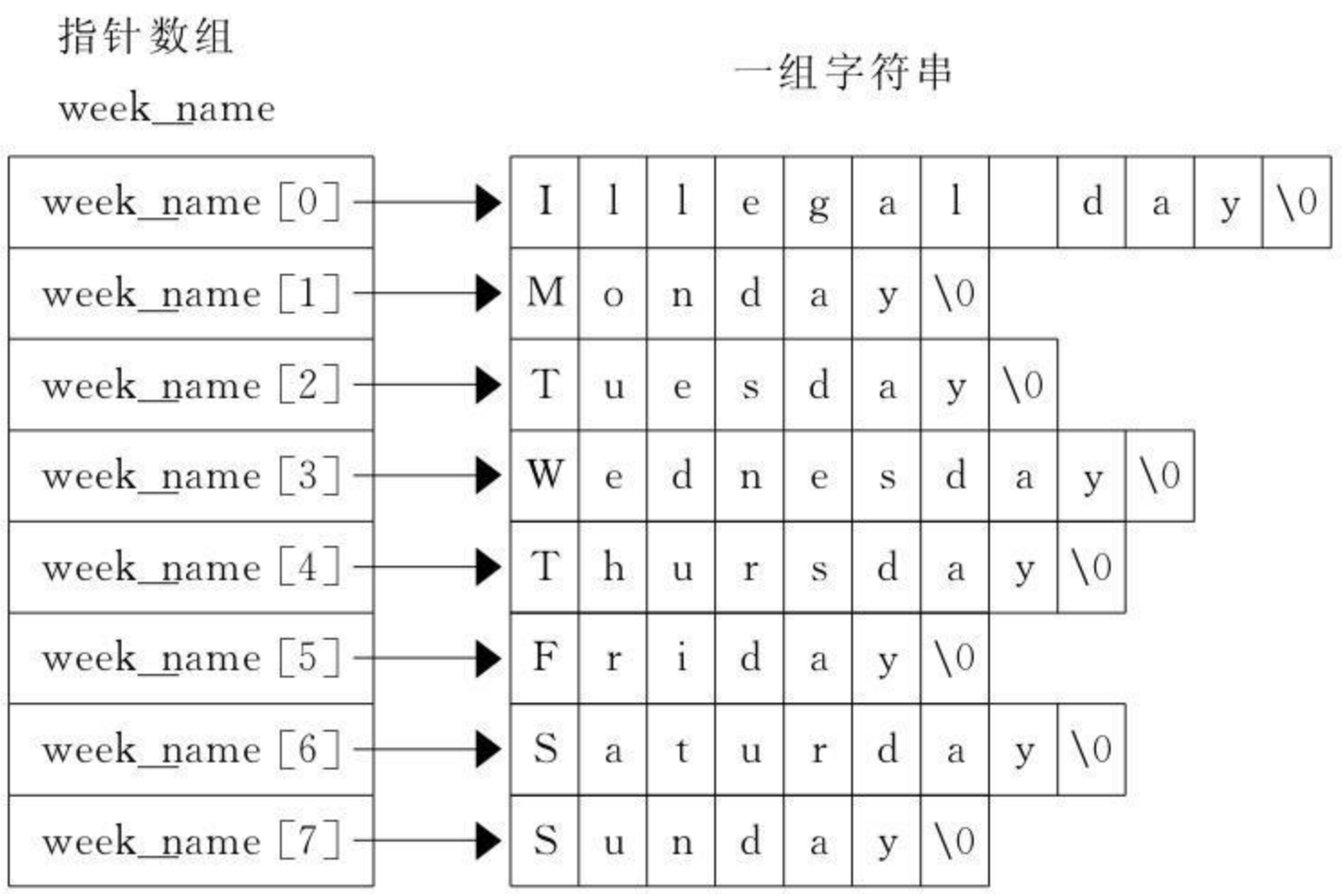


图7-30　字符串的存放形式

程序运行结果，如图7-31所示。

图7-31　例7.11程序运行结果

7.3.2 指向指针的指针

如果一个指针变量存放的又是另一个指针变量的地址，则称这个指针变量为指向指针的指针变量。

在前面已经介绍过，通过指针访问变量称为间接访问。由于指针变量直接指向变量，所以称为“单级间址”。而如果通过指向指针的指针变量来访问变量则构成“二级间址”，如图7-32所示。

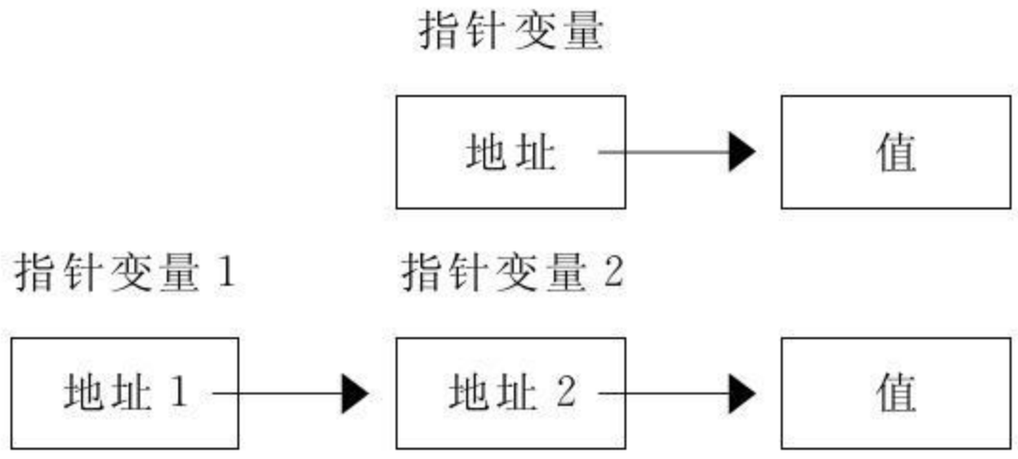


图7-32 二级地址

指向指针的指针变量说明的一般形式为：

类型说明符 * *指针变量名

例如：

```
int  a=5, *pa=&a;
int  **p;
```

p前面有两个*号，相当于*(*p)。显然*p是指针变量的定义形式，如果没有最前面的*，那就是定义了一个指向整型数据的指针变量。现在它前面又有一个*号，表示指针变量p是指向一个整型指针型变量的。

从而下面的程序行是正确的：

```
p=&pa;
```

这样，*p就是p所指向的另一个指针变量pa，如图7-33所示。

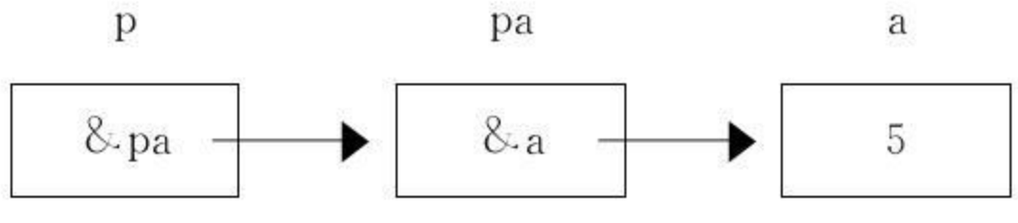


图7-33 指向指针的指针

【例7.12】 使用指向指针的指针变量输出一组字符串。

```
#include <stdio.h>
void main(void)
{ char *name[]={ "Wang Hai",
                 "Chen Chunguang",
                 "Li Ming",
                 "Jang Jian",
                 "Zhang Linlin"
               };
```

```
    char * *p;
    int  i;
    for(i=0;i<5;i++)
    { p=name+i;
      printf("%s\n",*p);
    }
}
```

【解析】 第3行到第7行定义了一个基类型为字符的指针数组name,其中的5个元素分别指向了5个字符串,第8行定义了一个基类型为字符的指向指针的指针变量p。通过p=name+i;使指针变量p指向指针数组name的各个元素,再通过*p指针输出各个字符串。如图7-34所示。

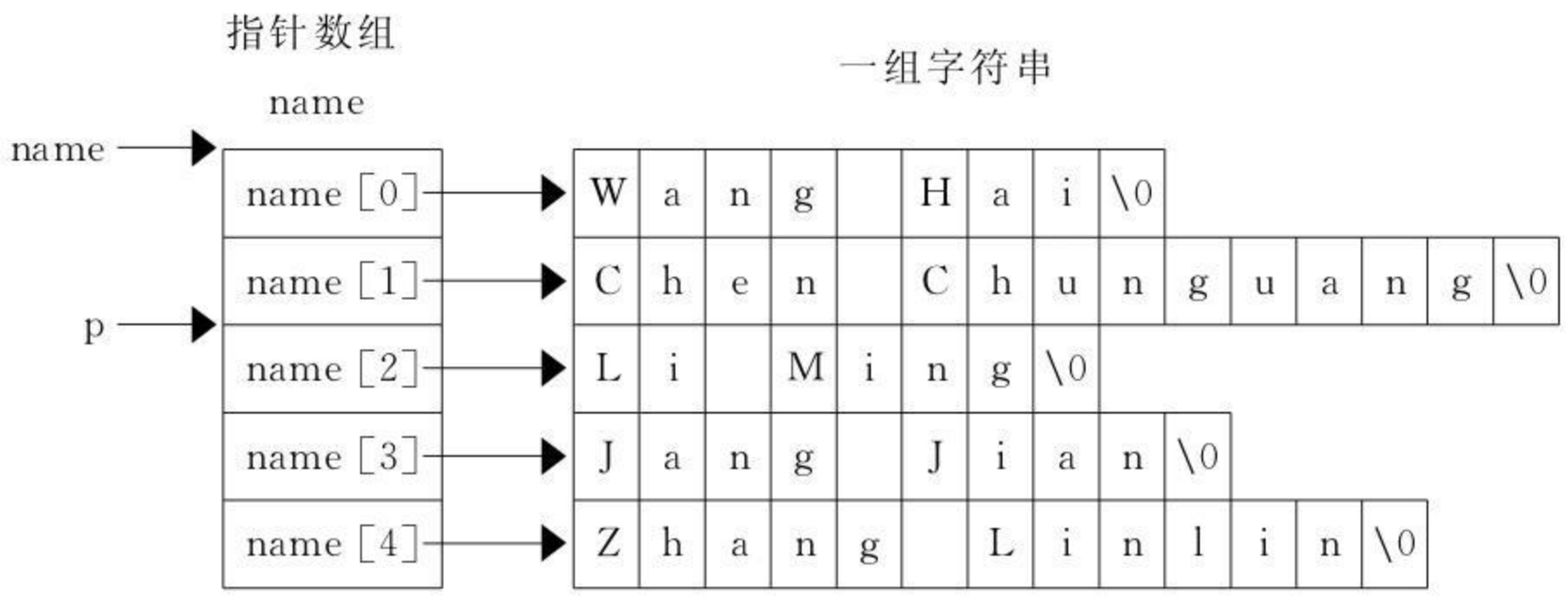


图7-34　各字符串的存放形式

程序运行结果,如图7-35所示。

图7-35　例7.12程序运行结果

【例7.13】 使用二级指针变量输出数组元素。

```
#include <stdio.h>
void main(void)
{ int a[5]={1,3,5,7,9};
  int *num[5];
  int **p,i;
  for(i=0;i<5;i++)
    num[i]=&a[i];
  p=num;
```

```
    for(i=0;i<5;i++)
    { printf("%d\t", * *p); p++;
    }
}
```

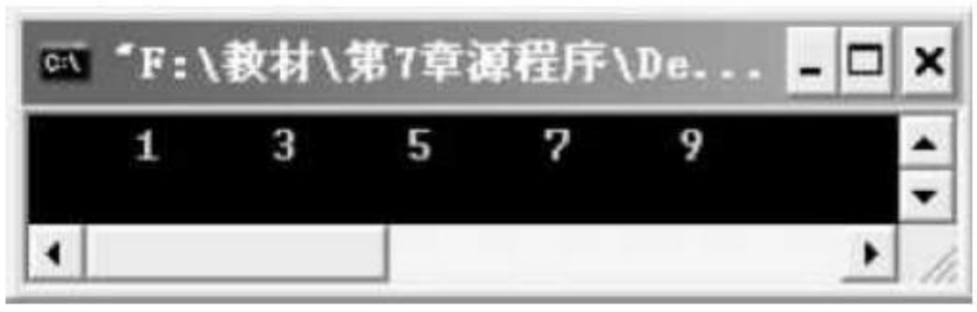


图 7-36 例 7.13 程序运行结果

程序运行结果,如图 7-36 所示。

【解析】 第 10 行的 * *p 前两个 * * 是间接访问运算符,* *p 等价于 *(*p)。而第 5 行的 * *p 前两个 * * 是说明符,说明 p 是指向指针的指针变量。

7.4 例题精解

【例 7.14】 输入 a 和 b 两个整数,并按从大到小的顺序输出。

```
#ineluele <stdio.h>
void main(void)
{ int *p1,*p2,*p,a,b;
  scanf("%d,%d",&a,&b);
  p1=&a;p2=&b;
  if(a<b)
  { p=p1;p1=p2;p2=p;
  }
  printf("\na=%d,b=%d\n",a,b);
  printf("max=%d,min=%d\n", *p1, *p2);
}
```

图 7-37 例 7.14 程序运行结果

程序运行结果,如图 7-37 所示。

【解析】 程序中的第 4 行使指针 p1,p2 分别指向了 a 和 b,第 5、6、7 行的条件语句的作用是当 a<b 时交换两个指针变量(p1 和 p2)的值,使 p1 指向 b(较大),p2 指向 a(较小);如 a>=b 时因不满足条件不会执行其中的语句。因此在执行了第 5、6、7 行条件语句后,指针 p1 一定指向较大的一个,p2 一定指向较小的一个。从而第 9 行输出的第 1 个量是较大的,第 2 个量是较小的。要注意的是无论在什么情况下 a 和 b 的值都没有交换,交换的只可能是 p1 和 p2 值。这从第 8 行的执行结果可以看到。

请同学们利用指针设计出等效的程序,并上机调试。

【例 7.15】 在输入的字符串中查找有无'k'字符。

```
#include <stdio.h>
void main(void)
{ char st[50], *ps;
  int i;
  ps=st;
  printf("input a string:\n");
  scanf("%s",ps);
```

```
for(i=0;ps[i]!='\0';i++)
  if(ps[i]=='k')
  { printf("there is a 'k' in the string\n");
    break;
  }
if (ps[i]=='\0')
  printf("There is no 'k' in the string\n");
}
```

程序运行结果,如图 7-38 所示。

图 7-38　例 7.15 程序运行结果

【解析】 程序完成了两项工作:一是利用第 7 行从键盘输入一个字符串通过指针 ps 存入字符数组 st ;二是利用循环使指针 ps 在字符串的移动,查找字符串中有无'k'字符,若有结束循环。否则直到遇字符串结束标志'\0'结束循环。最后通过条件语句来判断指针 ps 是否指向字符串结束标志'\0',若是表明字符串中没有字符'k'。

该题也可不用指针,直接用字符数组会更简单,这里主要是为了帮助同学们更进一步认识和理解指针,便于后续内容的学习。

【例 7.16】 要求把一个字符串的内容复制到另一个字符串中,并且不能使用 strcpy 函数。

```
#include <stdio.h>
void main(void)
{ char a[20]="I love China!",b[20];
  int i;
  for(i=0;*(a+i)!='\0';i++)
    *(b+i)=*(a+i);
  *(b+i)='\0';
  printf("string a=%s\nstring b=%s\n",a,b);
}
```

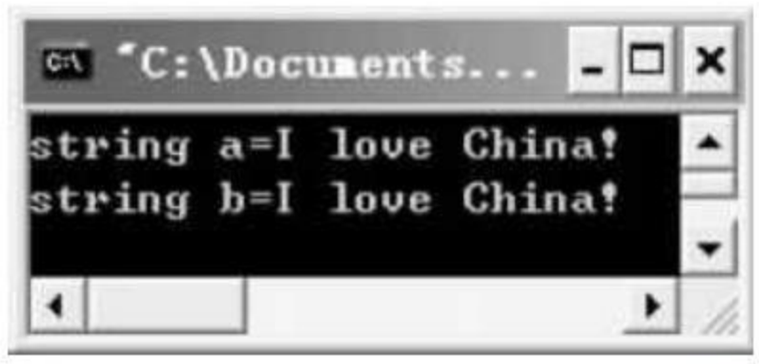


图 7-39　例 7.16 程序运行结果

程序运行结果,如图 7-39 所示。

【解析】 在本例中,程序完成了两项工作:一是把源字符串 a 中字符复制到目标字符串 b 中,二是判断所复制的字符是否为'\0',若是则表明源字符串结束,不再循环。否则,指针下移 1,指向下一字符。

分析还可发现我们可把指针的移动和赋值合并在一个语句中。进一步分析还可发现'\0'的 ASCⅡ码为 0,对于 while 语句只看表达式的值为非 0 就循环,为 0 则结束循环,因此

也可省去“!='\0'”这一判断部分而写为以下形式：

```
for(i=0; *(a+i);i++);
```

表达式的意义可解释为，源字符向目标字符赋值，移动指针，若所赋值为非0则循环，否则结束循环。如使用字符指针程序将更加简洁。

【例7.17】 简化后的程序如下所示。

```
#include <stdio.h>
void main(void)
{ char a[]="I love China!",b[20], *pb, *pa;
  pa=a; pb=b;
  while( *pb++= *pa++) ;
  printf("string a=%s\nstring b=%s\n",a,b);
}
```

程序运行结果，如图7-40所示。

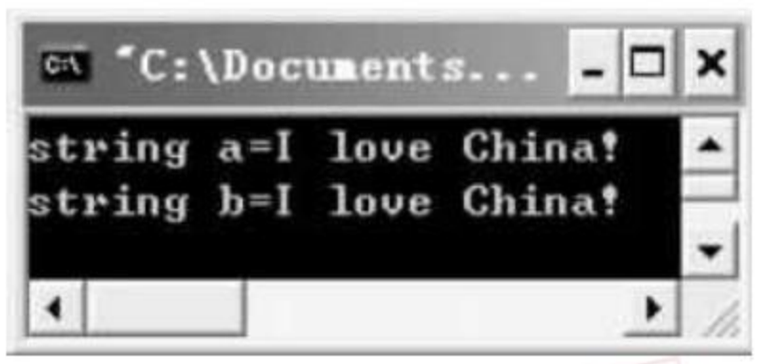


图7-40 例7.16程序运行结果

7.5 本章小结

指针是C语言最显著的特征之一，利用指针可以有效的使用数组，能方便地使用字符串，能在调用函数时得到多个返回值，能有效表示复杂的数据结构(如链表等)，能直接处理内存地址。正确巧妙地使用指针可以使程序更为简洁、紧凑。

指针变量也需要占用内存单元，编译程序也为指针变量分配存储空间。指针变量的值是另一个变量的地址，在定义一个指针变量而未初始化时，其值是不确定的，没有具体的意义。指针变量在使用前，必须使它指向另一个同种数据类型的变量。

通常使用指针时，必须按照下面3个步骤进行：

(1)说明指针及同类型的对象；

(2)指针指向所要访问的对象；

(3)通过指针引用对象。

指针与数组有密切的联系，用指针可以有效、方便地的使用数组。用数组名和用指针指向数组的区别：

(1)一个指针变量可以指向一个数组，也可以指向其他数组，但是一个数组名永远只能指向它所代表的数组首地址(编译后是一个固定的地址值)，不能指向其他任何地方。

(2)可以对指针变量进行赋值操作，但不能对数组名赋值，数组名是一个指针(即地址)，但不是指针变量。

(3)编译系统要为指针变量分配存储单元,但不为数组名分配存储空间,只为数组的每个元素分配存储空间。

(4)指针变量可以进行算术运算,但对数组名是不允许进行算术运算的。

习　　题

一、选择题

1. 已知:int a, * p;则正确的赋值表达式是_____。

A)p=&a　　B)p=a

C) * p=&a　　D) * p= * a

2. 对于基类型相同的指针变量,不能进行的运算是_____。

A)+　　B)-　　C)=　　D)!=

3. 已知:double * p[6];则它的含义是_____。

A)p 是指向 double 型变量的指针　　B)p 是 double 型数组

C)p 是指针数组　　D)p 是数组指针

4. 已知:char s[10], * p=s;则在下列语句中,错误的语句是_____。

A)p=s+5;　　B)s=p+5;

C)s[2]=p[4];　　D) * p=s[0];

5. 已知:char b[5], * p=b 则正确的赋值语句是_____。

A)b="abcd";　　B) * b="abcd";

C)p="abcd";　　D) * p="abcd";

6. 已知:char * * s;正确的语句是_____。

A)s="computer";　　B) * s="computer";

C) * * s="computer";　　D) * s='A';

7. 定义 int i, * p=&i 后,以下不正确的语句是_____。

A)i=p;　　B) * p=10;

C)i= * p;　　D) * p=2 * i+1;

8. 若有以下程序片段,请问执行后输出结果是_____。

```
int * * pp, * p,a=10;
p=&a; pp=&p;
printf("%d", * * pp);
```

A)10　　B)p 的地址　　C)a 的地址　　D)pp 的地址

9. 若有定义语句 int a[10]={0,1,2,3,4,5,6,7,8,9}, * p=a;则不是对 a 数组元素的正确引用是_____。

A)a[p-a]　　B) * (&a[i])

C) * p　　D) * (* (a+i))

10. 若有以下的定义:int a[]={1,2,3,4,5,6,7,8,9,10}, * p=a;则值为 3 的表达式是_____。

A)p+=2, * (p++);　　B)p+=2, * ++p;

C)p+=3, * p++;　　D)p+=2, ++ * p;

11. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int a=2,b=4,c=6;
  int *pa=&a,*pb=&b,*p;
  *(p=&c)=*pa*(*pb);
  printf("%d\n",c);
}
```

A)4　　B)6　　C)8　　D)10

12. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int a[]={2,4,6,8,10},b=1,i,*p;
  p=&a[1];
  for(i=0;i<3;i++)
    b+=*(p+i);
  printf("%d\n",b);
}
```

A)17　　B)18

C)19　　D)20

二、填空题

1. “*”称为_____运算符,“&”称为_____运算符。

2. 若两个指针变量指向同一个数组的不同元素,可能进行减法运算、_____运算和_____运算。

3. 设 int a[10],*p=a;则对 a[3]的引用可以是 p[_____]和 *(p _____)。

4. 若 d 是已定义的双精度变量,再定义一个指向 d 的指针变量 p 的语句是_____。

5. 设有 char *a="ABCD",则 printf("%s",a)的输出是_____;而 printf("%c",*a)的输出是_____。

6. 设有以下定义和语句,则*(*(p+2)+1)的值为_____。

```
int a[3][2]={10,20,30,40,50,60},(*p)[2];
p=a;
```

7. 若有如下定义和语句,则输出结果是_____。

```
int **pp,*p,a=10,b=20;
pp=&p;p=&a;p=&b;
printf("%d,%d\n",*p,**pp);
```

8. 执行下面程序段后,*ip 等于_____。

```
int a[5]={1,3,5,7,9},*ip;
ip=a;
ip++;
```

9. 设有变量定义：

```
int a[10]={1,2,3,4,5,6,7};
int *p=a;
```

则表达式*++p的值为_____。

10. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main(void)
{ int x[]={2,4,6,8},*p=&x[0],a=8,b,c;
  for(b=0;b<3;b++)
    c=(*(p+b)<a)? *(p+b):a;
  printf("%d\n",c);
}
```

三、应用题

1. 分析下面程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{ char a[]="program",*prt;
  for(prt=a;prt<a+7;prt+=2)
    putchar(*prt);
}
```

2. 分析下面程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{ char a[]="language",b[]="program";
  char *prt1=a,*prt2=b;
  int k;
  for(k=0;k<7;k++)
    if(*(prt1+k)==*(prt2+k))
      printf("%c",*(prt1+k));
}
```

3. 分析下面程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{ int a[2][3]={{1,2,3},{4,5,6}};
  int m,*prt;
  prt=&a[0][0];
  m=(*prt*(*prt+2)*(*prt+4));
  printf("%d\n",m);
}
```

4. 分析下面程序,写出运行结果。

```
#include <stdio.h>
char b[ ]="program";
char *a="PROGRAM";
void main(void)
{ int i=0;
  printf("%c%s\n",*a,b+1);
  while(putchar(*(a+i)))
    i++;
  printf("i=%d\n",i);
  while(--i)
    putchar(*(b+i));
  printf("\n%s\n",&b[3]);
}
```

四、编写程序题

1. 编写一程序,实现在字符串 sl 中的指定位置 n 处插入字符串 s2。

2. 用指针编写比较两个字符串 s 和 t 的程序。要求 s<t 时输出-1,s=t 时输出 0,s>t 时输出 1。

3. 编写一程序,其功能是:对一个长度为 N 的字符串从其中第 K 个字符起,删去 M 个字符,组成长度为 N-M 的新字符串(其中 N、M<80,K<=N)。

4. 输入五个单词,请将它们按从小到大的顺序排列后输出。

5. 输入字符串,请分别统计字符串中所包含的各个不同的字符的数量。如:

输入字符串:abcedabcdcd

则输出:a=2 b=2 c=3 d=3 e=1

第8章　函　　数

【教学提示】

C语言程序可由一个主函数和若干个子函数构成，函数是程序构成的基本单位。本章主要介绍函数的结构、定义方法和调用方法；函数调用中数据传递的规则及应用；函数的嵌套调用和逆归调用；变量的作用域和存储类型；命令行参数等概念。

【核心概念】

函数的参数传递　函数的调用　函数的嵌套与递归调用　全局变量　局部变量　动态变量　静态变量　内部函数与外部函数。

8.1　函数的基本概念

8.1.1　概　述

函数这个名词我们前几章不断提到，如一个程序必有一个主函数 main()；标准输入/输出函数 scanf()/printf()等。可以说C语言中完成各种各样的功能都是通过不同函数来完成的。一个具备一定规模的C语言程序可由以下多个函数构成，如图8-1所示。

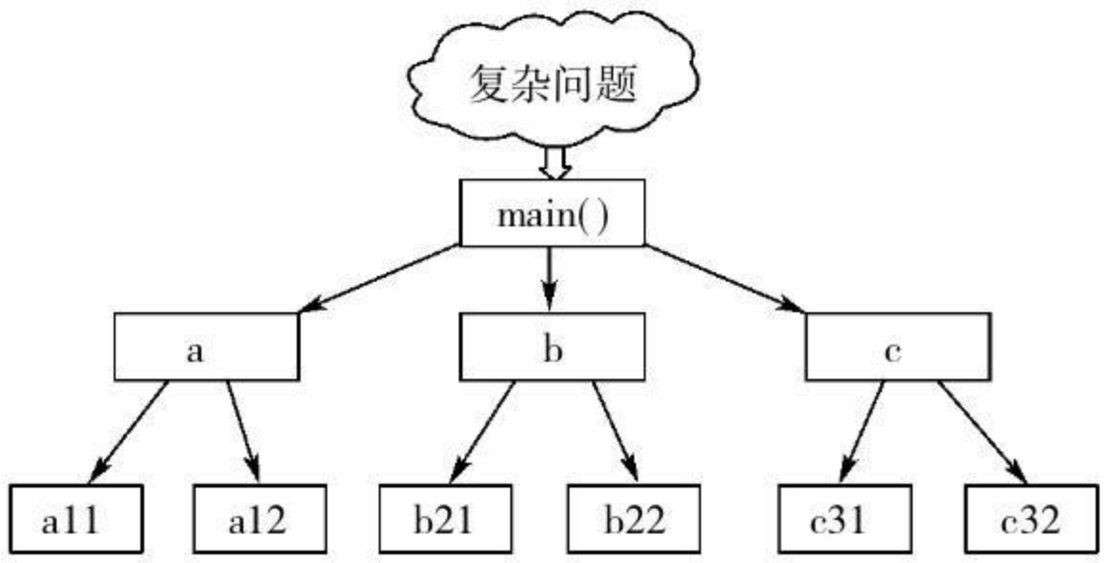


图8-1　C程序的构成

那么为什么要用函数呢？主要原因有以下几点：

1. 函数是能够完成一个特定功能的一个独立模块；

2. 便于结构化程序设计采用逐步求精的方法，将一个较大的程序分解为若干个较小程序模块(即函数)来实现；

3. 主函数可以调用其他函数，其他函数之间也可以互相调用，减少程序书写的冗余，增强程序的可读性等。

C语言中的函数可按不同的角度分类：

按函数定义分{库函数：由系统提供用户无须定义和说明就可以直接调用它们。
用户定义函数：由用户按需要编写的专门用于实现特定功能的函数。

按功能分{有返回值函数：此类被调用执行后，将向调用者返回一个函数值。
无返回值函数：此类函数用来完成特定的功能，执行完后不向调用者返回函数值。

按使用范围分{内部函数：只能在本源文件中使用。
外部函数：可在整个源程序中使用。

按数据传送分{无参函数：函数定义、函数说明及函数调用都不带参数。
有参函数：函数调用时要有参数。

8.1.2 函数的定义方法

函数定义的一般形式为：

类型说明符 函数名([形式参数表])

{ 说明部分

语句部分

}

关于函数定义一般形式的几点说明：

(1)中括号[ ]表示可选项。

(2)类型说明符定义了函数返回值的类型，它可以是任何一种有效的数据类型。若无类型说明符，则默认为整型。

(3)函数名是唯一标识一个函数的名字，它的命名规则和标识符命名规则完全相同。

(4)形式参数表是用来在主调函数和被调函数之间进行数据传递的载体。它是以逗号分隔的前面带有类型说明符的变量名表，若形参为空，则为无参函数，但函数名后面的圆括号不能省略。

(5)花括号括起来的部分称为函数体。说明部分主要对函数内使用的变量和所调用的函数的类型进行说明。语句部分是用来完成函数具体功能的语句。

【例 8.1】 定义一个函数，求两个整数的最大值。

```
int imax(int a,int b)
{ int c;                    /*说明部分*/
  c=a>=b? a:b;              /*语句部分*/
  return c;
}
```

说明：自定义一个函数名为 imax；函数的返回值类型为 int 型；形参表里有两个整型形参 a,b，注意它们的书写形式，与变量说明不同；花括号括起来的为函数体。

8.2 函数参数和函数的值

8.2.1 形式参数和实际参数

函数的参数分为形参和实参两种。在函数定义时，函数名后面圆括号内的参数称为形式参数，简称为形参。在函数调用时，函数名后面圆括号内的参数称为实际参数，简称实参。

形参和实参的功能是数据传送，当函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送。

【例8.2】 定义一个函数，求两个整数的最大值。在主函数中输入两个数据，调用定义函数求其最大值并输出。

```
#include <stdio.h>
int imax(int a,int b)
{ int c;
  c=a>=b? a:b;
  return c;
}
void main(void)
{ int x,y,z;
  printf("Please enter two integer:\n");
  scanf("%d %d",&x,&y);
  z=imax(x,y);
  printf("Max number is %d\n",z);
}
```

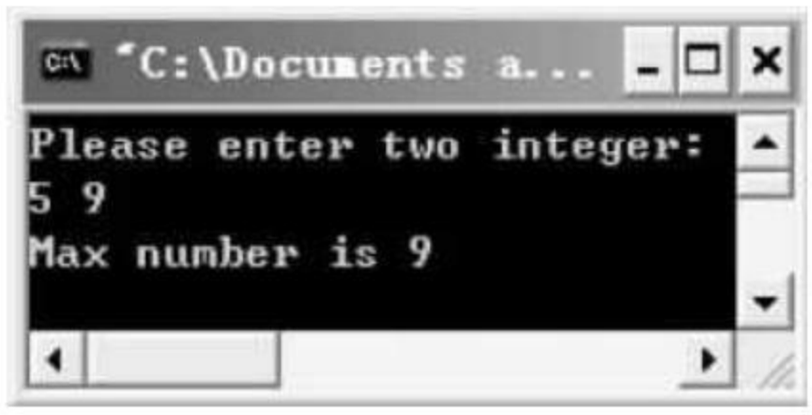


图8-2 例8.2程序运行结果

程序运行结果，如图8-2所示。

程序里imax()函数中的(a,b)为形式参数；main()函数调用imax()函数中的(x,y)为实际参数。形参通过实参调用得到数据值，从而函数完成数据处理过程。

关于形参和实参的几点说明：

(1) 形参变量只有在被调用时才分配内存单元，被分配的存储单元与实参不是同一个存储单元，在调用结束时释放所分配的内存单元。就是说形参只有在函数内有效，调用结束后主调函数不能再使用该形参。上例设x=5，y=9，调用过程如图8-3所示。

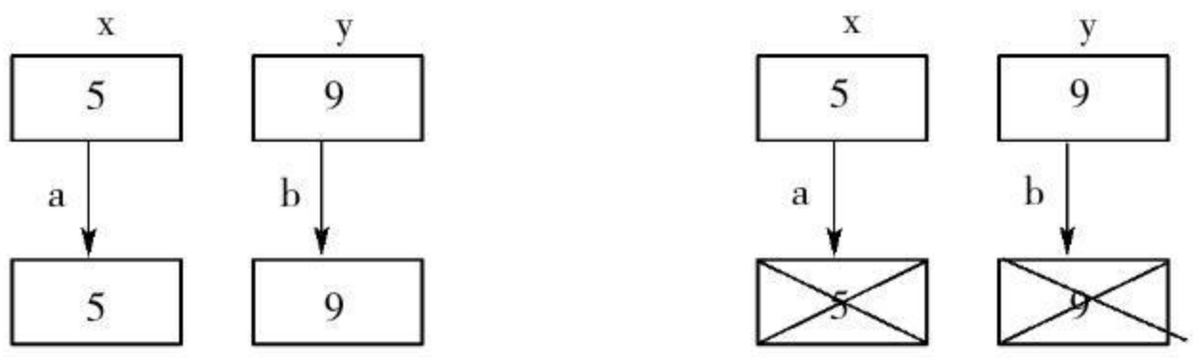


图8-3 函数调用示意图

(2) 实参在函数调用时必须是一个有确定值的量，它可以是常量、变量、表达式及函数等。

(3) 实参向形参的数据传递是一种单向的值传递，即实参的值只能传递给形参，而形参的值不能传递给实参。

(4) 实参和形参在数量、类型、顺序上应严格一一对应。

8.2.2 函数的返回值

函数的返回值是指函数被调用、执行后，返回给主调函数的值。函数的返回值是通过

return 语句实现的。

return 语句的一般形式为：

return 表达式；或 return（表达式）；

说明：

(1)一个函数体中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，最多只能返回一个函数值。

(2)函数的返回值类型要求跟函数的类型说明符类型兼容，最好一致，否则，自动转换为函数的类型说明符类型。

(3)一个函数既可以有返回值，也可以无返回值。对于无返回值的函数，其函数类型说明符应明确定义为“void”型。

8.3 函数的调用

8.3.1 函数调用的形式与方式

函数调用是指主调函数中调用函数的形式和方法。C 语言中是通过调用函数来执行函数体，实现特定功能的。

1. 函数调用形式

函数调用的一般形式为：

函数名(实际参数表)

如果调用有参函数，实参表中各参数用逗号分隔，如 c＝imax(a,b)；如果调用无参函数，则实参表为空，如 c＝imax()。

2. 函数调用方式

函数调用方式可分为以下三种：

(1)函数语句

把函数调用作为一个语句。主要用于无返回值或不需要通过返回值传递数据的函数调用。例如：

```
imax(a,b);
```

这时不要求函数带回值，只要求函数完成一定的操作。

(2)函数表达式

函数作为表达式的一部分，且函数返回值参与表达式的计算。例如：

```
y＝2 * imax(a,b);/ * 该语句把函数的值乘以 2 后赋值给 y * /。
```

(3)函数参数

函数调用放在另一调用函数的实参表中，以该函数返回值作为一个实参。例如：

```
d＝imax(x,imax(y,z));
```

先求出 imax(y,z)的值作为 imax(x,imax(y,z))的一个实参参与运算。

8.3.2 被调函数的声明和函数原型

被调函数的声明和函数原型一般是对用户自定义函数而言的，而系统提供的标准函数

则不需要再声明。

【例 8.3】　求最大值实例。

```
1   #include <stdio.h>
2   void main(void)
3   { int x,y,z;
4     int imax(int,int);              /*被调函数的原型声明*/
5     printf("Please enter two integer:\n");
6     scanf("%d %d",&x,&y);
7     z=imax(x,y);
8     printf("Max number is %d\n",z);
9   }
10  int imax(int a,int b)
11 { int c;
12   c=a>=b? a:b;
13   return c;
14 }
```

上例程序中第 4 行就是对被调用函数 imax 的说明，而 printf、scanf 则不需说明。第 10 行至 14 行为自定义函数体。

一个函数调用了程序中定义的其他函数，且主调函数与被调函数在同一个程序文件中，一般应在主调函数定义之前或在调用语句之前，对被调函数作原型声明。

1. 原型声明的一般形式

类型说明符　函数名(类型说明符,类型说明符,……);

或 类型说明符　函数名(类型说明符　形参名,类型说明符　形参名,……);

2. 说明

(1)当被调函数定义出现在主调函数之前，则可以不必对被调函数作说明。如【例 8.2】中主调函数中没有对被调函数的说明。

(2)当被调函数的返回值为整型时，可不对被调函数加以说明而直接调用(这是对 TC 环境而言，但在 VC 环境下，不管什么类型都要进行原型声明)。

(3)被调函数的说明可以放在所有函数定义之前，在函数外预先说明函数的类型，在以后的各主调函数中，可不再对被调函数作说明。如【例 8.3】中第 4 行可以调到第 2 行前，也是正确的。

(4)对系统提供的库函数，一般在程序文件头用“#include”包含进去，如调用 getchar() 在程序前用“#include <stdio.h>”。

8.4　数组作为函数参数

有参函数调用时，需要由实参向形参传递参数。函数参数在传递数据时可以采用两种方式：一种传值方法；另一种传址方法。当数组元素作为参数传递时，如同变量传递，采用传值方法；而数组名作为参数传递时，采用传址方法。

1. 数组元素作为函数实参

它与变量作为实参一样，是单向值传递方式。

【例 8.4】 有一整型数组，判断各元素的正负，若为非负输出值为 1，否则输出 0。

```
#include <stdio.h>
void judge(int);          /* 函数原型声明 */
void main(void)
{ int i,x[6];
  printf("Please enter 6 integers\n");
  for(i=0;i<6;i++)
  { scanf("%d",&x[i]);
    judge(x[i]);
  }
}
void judge(int a)
{
  printf("%d",(a>=0)? 1:0));
}
```

图 8-4 例 8.4 程序运行结果

程序运行结果，如图 8-4 所示。

程序中定义的 judge 函数是用来完成整数正负判断的，主函数在 for 语句中每输入一个值，调用一次 judge 函数，并且 judge 的实参为键盘输入的数组元素值。

2. 数组名作为函数参数

数组名作为函数参数时，实参和形参都应是数组名或数组指针；数组名作实参不是把数组中所有值传递给形参，而是把实参数组的起始地址传递给形参数组名，使形参数组和实参数组共占实参数组的那段内存单元。

【例 8.5】 输入 10 个学生的 C 语言成绩至一维数组 score，编一函数求其平均成绩。

```
#include <stdio.h>
double aver(float a[]);
void main(void)
{ float score[];int i;
  for(i=0;i<10;i++)
    scanf("%f",&score[i]);
  printf("Average score is %5.1f\n",aver(score));
}
double aver(float a[ ])
{ int i;double sum=0.0;
  for(i=0;i<10;i++)
    sum+=a[i];
  return sum/10.0;
}
```

程序运行结果，如图 8-5 所示。

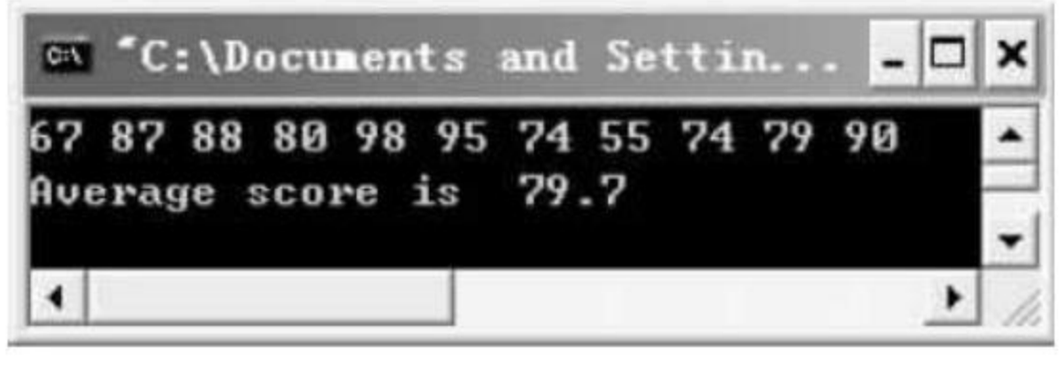


图 8-5　例 8.5 程序运行结果

程序中定义了一个求平均成绩的函数 aver，它的形参是一个实型数组，在主调函数中 aver(score)就用了一个已定义的数组名 score 作为实参来完成数据处理。

说明：

(1)用数组名作函数参数，应该在主调函数和被调函数中分别定义数组。

(2)形参数组和实参数组的类型必须一致，否则将引起错误。

(3)数组名作参数时，形参和实参实际上是同一个数组。在调用函数过程中改变了形参的数组元素的值实际上就是改变了主调函数中实参数组元素的值。

(4)在被调函数中声明形参数组可不指定数组大小，一般可另设一参数，传递数组的大小。

如【例 8.5】可以改写为以下程序，分别求 5 个同学和 10 个同学的平均成绩。

```
#include <stdio.h>
float aver(float a[],int);
void main(void)
{ float score[10];int i,n;
  for(i=0;i<10;i++)
    scanf("%f",&score[i]);
  printf("Average score is %5.1f",aver(score,5));   /*求 5 个学生的平均分*/
  printf("Average score is %5.1f",aver(score,10));   /*求 10 个学生的平均分*/
}
float aver(float a[],int n)
{ int i;float sum=0.0;
  for(i=0;i<n;i++)
    sum+=a[i];
  return sum/n;
}
```

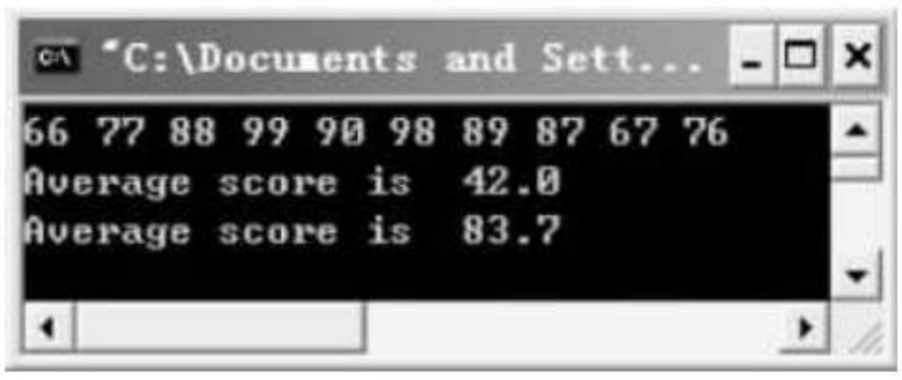


图 8-6　例 8.5 程序运行结果

程序运行结果，如图 8-6 所示。

与【例 8.5】不同的是函数定义形参中多了一个整型变量 n，用来确定形参数组元素的个数，n 值不同，处理的数组元素的个数就不同。

(5)多维数组名也可作为函数的形参和实参。被调函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明，但第二维的大小不能省略。如：

【例 8.6】　给定一个二维数组 a[3][3]，求其转置。

```
#include <stdio.h>
#define N 3
void main(void)
{ void convert(int b[][N]);
  int i,j,a[N][N];
  for(i=0;i<N;i++)
     for(j=0;j<N;j++)
        scanf("%d",&a[i][j]);
  convert(a);
  printf("Converted array is:\n");
  for(i=0;i<N;i++)
    { for(j=0;j<N;j++)
       printf("%4d",a[i][j]);
       printf("\n");
    }
}
void convert(int b[][N])
{ int i,j,temp;
  for(i=0;i<N;i++)
  for(j=0;j<i;j++)
  { temp=b[i][j];
    b[i][j]=b[j][i];
    b[j][i]=temp;
  }
}
```

图 8-7 例 8.6 程序运行结果

程序中 convert(int b[N][N])与 convert(int b[][3])是等价的。程序运行结果,如图 8-7所示。

说明:数组名作函数参数传递的是地址值,也就是说把实参数组的首地址赋予形参数组名。形参数组名取得该首地址之后,也就等于有了实在的数组。实际上是形参数组和实参数组为同一数组,共同拥有一段内存空间。

8.5 指针作为函数参数

C语言对函数的调用采用单向值传递的方法,在被调用的函数中对形参的任何修改都不会影响实参。

【例 8.7】 实现两数变换。

```
#include <stdio.h>
```

```
void exchange(int x,int y)
{ int temp;
  temp=x;
  x=y;
  y=temp;
}
void main(void)
{ int a=4,b=5;
  printf("a=%d,b=%d\n",a,b);
  exchange(a,b);
  printf("a=%d,b=%d\n",a,b);
}
```

图 8-8　例 8.7 程序运行结果

程序运行结果，如图 8-8 所示。

【解析】 函数定义者的本意是想通过 exchange(a,b)把 a,b 值进行交换的，可结果并没有完成用户的要求。原因是调用时实参 a 与形参 x 不占同一内存单元，实参 b 与形参 y 不占同一内存单元，交换只在 x,y 所占的内存单元中进行的，而并非 a,b 所占内存单元中进行的，调用结束 x,y 所占内存单元释放，a,b 所占内存单元值保持不变。

因此，要想保持被调函数中变化了的值带出主调函数，我们得想法使实参和形参都指向同一内存单元。

我们的方法是：对于一批数据可用上节所讲的用数组名（本质上讲就是指针）作为函数参数，如例 8.6；对于少数几个数据可用指针变量作为函数参数，如把上例改为：

【例 8.8】 用指针作为函数参数，实现两数交换。

```
#include <stdio.h>
void exchange(int *x,int *y)
{ int temp;
  temp=*x;
  *x=*y;
  *y=temp;
}
void main(void)
{ int a=4,b=5;
  printf("a=%d,b=%d\n",a,b);
  exchange(&a,&b);
  printf("a=%d,b=%d\n",a,b);
}
```

图 8-9　例 8.8 程序运行结果

程序运行结果，如图 8-9 所示。

【解析】 在函数调用过程中实参 &a 与形参 x 都是指向同一内存单元变量 a 的地址，实参 &b 与形参 y 都是指向同一内存单元变量 b 的地址，而在调用函数的函数体内，完成了内存单元内容的交换，即对主调函数中变量 a,b 的修改。当调用结束后 x,y 被分配指向 a,

b 内存单元的地址空间被释放，改变了的内存单元内容 a，b 值仍然保存，即 a 为 5，b 为 4。

8.6 嵌套调用与递归调用

8.6.1 嵌套调用

C 语言中函数定义是互相平行、独立的，也就是说 C 语言不支持函数的嵌套定义，即在定义一个函数时不能在函数体内再定义另一个函数。但 C 语言支持函数的嵌套调用，如图 8－10 所示。

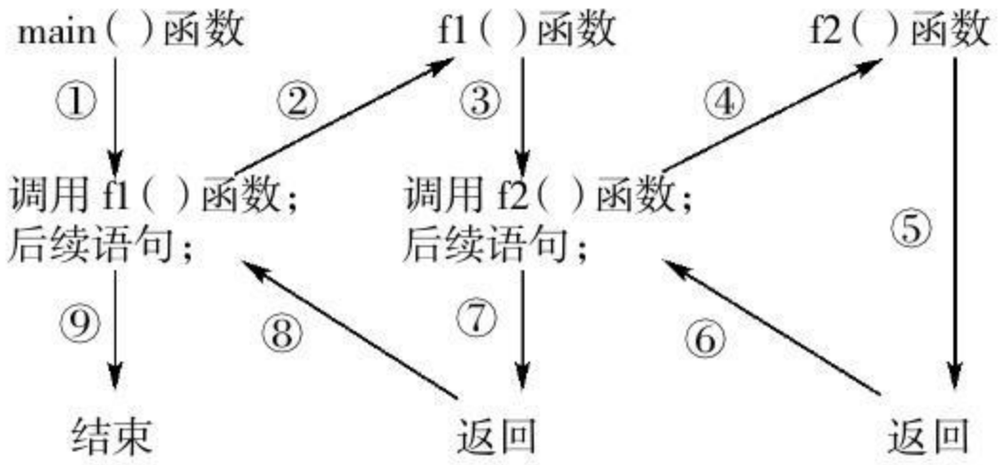


图 8－10 函数的嵌套调用

在主函数中遇到调用 f1 函数就沿着箭头所指方向去执行 f1 函数体，在 f1 函数中又遇到调用 f2 函数，就沿着箭头所指方向去执行 f2 函数体，f2 函数执行完毕又返回到 f1 函数调用 f2 函数的断点，再执行 f1 函数后续语句，f1 函数执行完后再返回到主函数调用 f1 函数的断点，继续执行后续语句，整个调用执行过程见上图。

【例 8.9】 用函数调用求一元二次方程的根。

要求：

① main()函数中输入 a、b、c；

② 调用求根 root 函数求方程的两个根；

③ root 函数调用 delta 函数求根的判别式。函数调用过程，如图 8－11 所示。

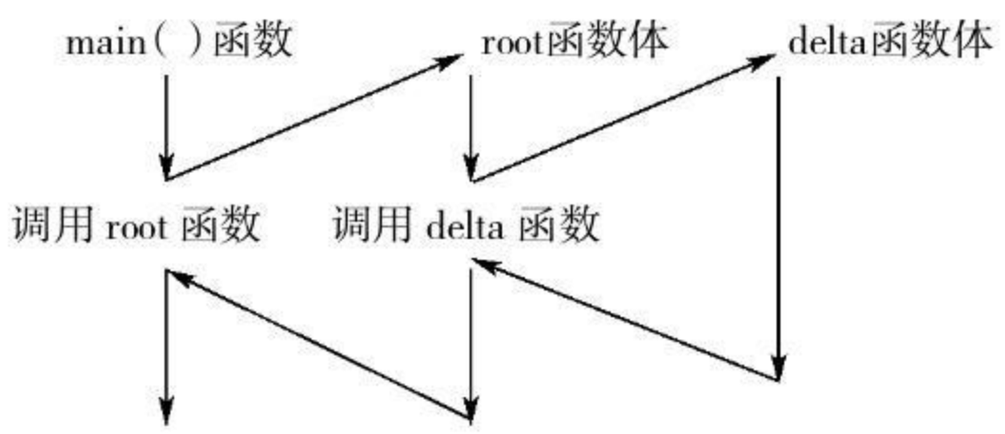


图 8－11 求一元二次方程的根的调用过程

```
#include <stdio.h>
#include <math.h>
double delta(double a2,double b2,double c2)
{ double data;
  data=b2 * b2-4 * a2 * c2;
  return data;
```

```
}
void root(double a1,double b1,double c1)
{ double x1,x2,realpart,imagepart;
  if (a1<=1e-6)          /*实数里没有精确的 0 值,这里用 1e-6 取代*/
    { printf("不是一元二次方程!");
      return;
    }
  else
  if (fabs(delta(a1,b1,c1))<=1e-6)   /* 这里用 1e-6 取代 0*/
  printf("有两个相等实根:%6.2f",-b1/(2*a1));
  else
  if (delta(a1,b1,c1)>1e-6)
    { x1=(-b1+sqrt(delta(a1,b1,c1)))/(2*a1);
      x2=(-b1-sqrt(delta(a1,b1,c1)))/(2*a1);
      printf("x1=%6.2f,x2=%6.2f\n",x1,x2);
    }
  else
  { realpart=-b1/(2*a1);          /*复数的实部*/
    imagepart=sqrt(-delta(a1,b1,c1))/(2*a1);   /*复数的虚部*/
      printf("x1=%6.2f+%6.2fi\n" ,realpart,imagepart);
      printf("x2=%6.2f-%6.2fi\n",realpart,imagepart);
    }
}
void main(void)
{ float a,b,c;
  scanf("%f %f %f",&a,&b,&c);
  root(a,b,c);
}
```

图 8-12　例 8.9 程序运行结果

若输入 1 2 3 ↙,程序运行结果,如图 8-12 所示。

程序在执行过程中,main 调用了 root,而 root 又调用了 delta,出现函数的嵌套调用关系。

8.6.2　递归调用

C 语言除了嵌套调用外,另一特点就是函数的递归调用,即一个函数在该函数体内可以直接或间接调用该函数本身。

【例 8.10】　用递归调用求 n!,求解公式 $n!=\begin{cases}1 & (n=0)\\ 1 & (n=1)\\ n*(n-1)! & (n>1)\end{cases}$

```
#include <stdio.h>
```

```
long fact(int i)
{ long f;
  if (i<0)
    printf("i<0,data error! \n");
  else
  if(i==0||i==1) f=1;
  else f=fact(i-1)*i;
  return f;
}
void main(void)
{ int n;long y;
  scanf("%d",&n);
  y=fact(n);
  printf("%d!=%ld\n",n,y);
}
```

图 8-13　例 8.10 程序运行结果

在调用函数中若输入 4 ↙,程序运行结果,如图 8-13 所示。

函数递归调用可理解为一种特殊的函数调用,如图 8-14 所示。

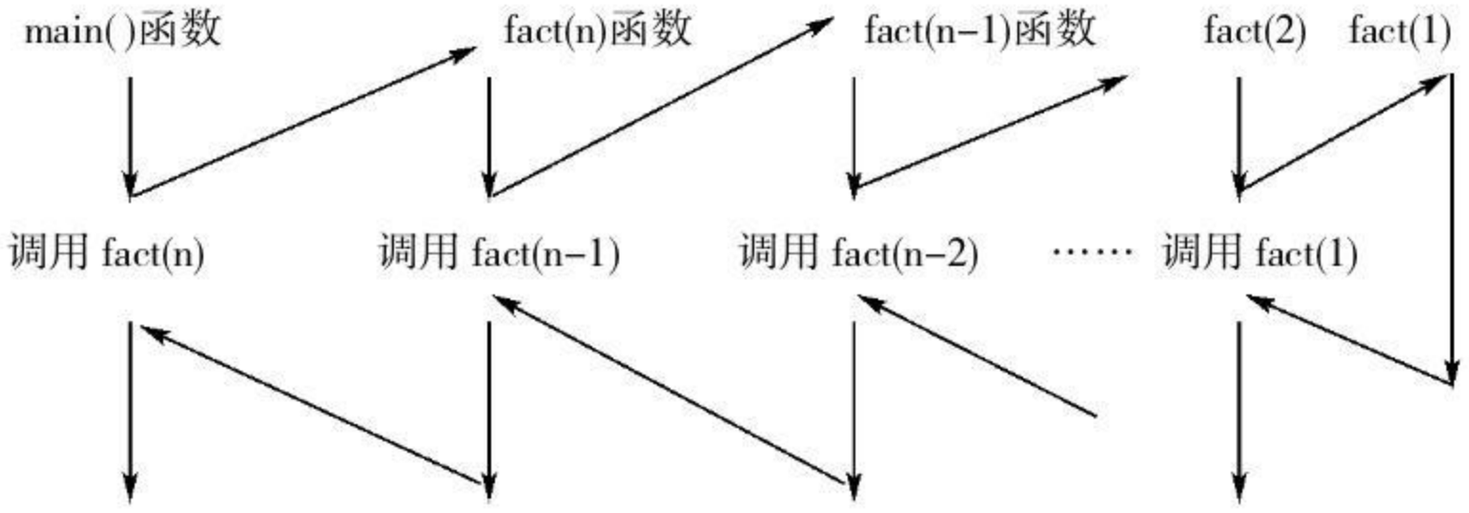


图 8-14　函数的递归调用

主函数 main( )调用 fact 函数,而 fact 函数不断地调用它自己 fact。注意 fact 函数递归调用时,第一,每调用一次参数表里的参数要修改,如上例中(n－1);第二,递归调用要有终止的时候,不能无休止。如上例中 if(i==0||i==1) f=1(即 f=fact(0)*1=1*1=1)就是结束函数继续调用的执行;第三,在最后一次调用结束返回时,就是不断累计求值的过程,如 f=fact(1)=1,f=fact(2)=fact(1)*2,f=fact(3)=fact(2)*3=2*3,f=fact(4)=fact(3)*4=6*4。

【例 8.11】 用函数递归调用求 Fibonacci 数列前 20 项的值。它的公式如下:

$$fib(n)=\begin{cases}1 & (n=1)\\ 1 & (n=2)\\ fib(n-1)+fib(n-2) & (n>2)\end{cases}$$

```
#include <stdio.h>
long fib(int n)
{ if (n==1)
```

```
    return 1;
  else if(n==2)
    return 1;
  else
    return fib(n-1)+fib(n-2);
}
void main(void)
{ int i;
    for(i=1;i<=20;i++)
  {
      printf("%8ld",fib(i));
      if(i%4==0)printf("\n");
  }
}
```

程序运行结果,如图 8-15 所示。

图 8-15　例 8.11 程序运行结果

8.7　存储类型

8.7.1　变量的存储类型

C 语言说明变量时给出两方面的信息:数据类型和存储类型。数据类型表示存储在变量中的数据格式;存储类型表示系统为变量分配存储空间的方式。C 语言的存储类型有:自动型(auto)、外部型(extern)、静态型(static)和寄存器型(register)。C 语言默认为自动型,可以省略不写,四种类型的作用域、生存期,见表 8-1 所列。

表 8-1　变量的存储类型

存储类型	自动型(auto)	外部型(extern)	静态型(static) 只限函数内部情况	寄存器型 (register)
出现范围	函数内部	任何可出现说明部分的位置	函数内部	函数内部

（续表）

存储类型	自动型(auto)	外部型(extern)	静态型(static) 只限函数内部情况	寄存器型 (register)
判别方法	1)在变量说明前出现 auto 2)在函数内部没有存储类型说明的变量	1)在变量说明前出现 extern 2)在函数外部没有存储类型说明的变量	在变量说明前出现 static	在变量说明前出现 register
作用域	说明该变量的函数内	出现说明位置开始直至程序正文结束	说明该变量的函数内	说明该变量的函数内
生存期	说明该变量的函数被调用时	整个程序执行期	整个程序执行期	说明该变量的函数被调用时
注意事项	不同的函数体内允许使用相同的变量而不会混淆	可在一个源程序中若干源文件使用	只赋初值一次，以后变化的值，没有修改始终存在	个数有限，常用来存放循环变量，提高程序执行速度

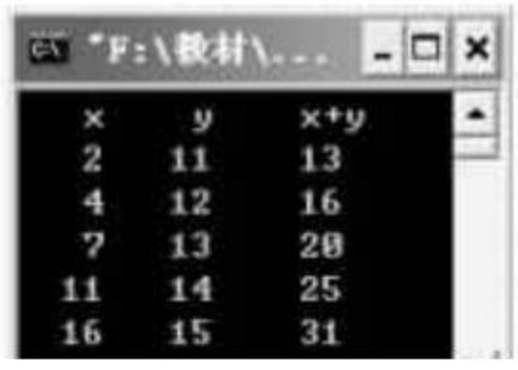
8.7.2 变量的作用域和生存期

变量的作用域是指变量在程序正文中合法出现的范围，分为局部变量和全局变量，是从空间的角度来划分的。生存期是指变量在程序中存在的时间长短，是从时间的角度来说的。两者有联系但不是同一回事，从上表中读者可以体会一下它们的异同。下面我们通过例题来说明变量的存储类型和作用域。

【例 8.12】 使用局部静态变量。

```
#include <stdio.h>
void main(void)
{ int ab(int,int);                          /*函数说明语句*/
  register int i;                           /*循环变量用寄存器类型*/
  printf("    x     y     x+y\n");
  for(i=0;i<5;i++)
    printf("%4d\n",ab(i+1,i+10));
}
int ab(int a,int b)                         /*函数定义开始*/
{ static int x=1;                           /*定义 x 为静态局部变量*/
  auto int   y=1;                           /*定义 y 为自动局部变量*/
  x+=a;
  y+=b;
  printf("%4d %4d",x,y);                    /*打印出 x,y 的值*/
  return x+y;                               /*返回 x+y 的值*/
}
```

程序运行结果，如图 8-16 所示。



对局部静态变量 x 和局部动态变量 y 以及 x+y 在程序中变化情况可通过表 8-2 表示。

表 8-2　局部静态变量和局部动态变量在程序中的变化

调用次数	调用时初值				调用结束时的值		
	x	y	a	b	x	y	x+y
第 1 次	1	1	1	10	2	11	13
第 2 次	2	1	2	11	4	12	16
第 3 次	4	1	3	12	7	13	20
第 4 次	7	1	4	13	11	14	25
第 5 次	11	1	5	14	16	15	31

从表中我们看出静态局部变量 x 的值只赋值一次 x=1，以后保存它改变后的值 2、4、7、11、16，生存期是整个程序执行期；而动态局部变量 y 的初值都为 1，调用结束的值 11、12、13、14、15 不保存，生存期是该变量的函数被调用时；寄存器变量 i 用法同动态局部变量 y。

【例 8.13】　全局变量和局部变量的使用。

```
#include <stdio.h>
extern int a=3,b=4;                    /*a,b 为外部全局变量,它的作用域到程序结束*/
int imax(int a,int b)                  /*形参 a,b 为局部变量作用域在该函数内*/
{ printf("a=%d,b=%d\n",a,b);
  return (a>b? a:b);
}
void main(void)
{ printf("a=%d,b=%d\n",a,b);
  { int a=10;                          /*a 为局部变量,作用域在该复合语句内有效*/
    printf("%d\n",imax(a,b));
  }
  printf("a=%d,b=%d\n",a,b);
}
```

图 8-17　例 8.13 程序运行结果

程序运行结果，如图 8-17 所示。

本程序变量都是 a、b，但它们的作用域是各不相同的，见程序中的说明和程序运行的结果。

重点：变量的存储类型与其作用域和生存期的关系

8.7.3　内部函数与外部函数

内部函数是指在一个源文件中定义的函数只能被本文件中的函数调用，而不能被同一源程序中其他文件的函数调用。其定义的一般形式为：

[static]　类型说明符　函数名(形参表)

如：static int max(int a,int b)

内部函数也称为静态函数。使用内部函数时，可以使函数的调用只局限于本文件，如果在不同的文件中有同名的内部函数名，则互不干扰。

外部函数是指在整个源程序中都可使用，其定义的一般形式为：

[extern] 类型说明符 函数名(形参表)

如：extern int max(int a,int b)

说明：

(1)中括号表示可选项，在函数定义时没有 extern 或 static，则默认为 extern。

(2)在一个源文件函数中调用其他源文件中定义的外部函数时，应用 extern 说明被调函数为外部函数。

【例 8.14】 用外部函数实现：输入一字符串，将其所有小写字母转换为大写字母。

```
file1.c(文件 1)                    /*文件 1 的名称,以下是文件的源程序*/
#include <stdio.h>
extern void upper(void);           /*说明本文件要用到文件 2 中的函数 upper()*/
char string[200];                  /*定义外部字符型一维数组*/
void main(void)
{ printf(" Please enter a string:\n");
  printf("%s\n",string);
  scanf("%s",string);
  upper();
  printf("%s\n",string);
}
file2.c(文件 2)                    /*文件 2 的名称,以下是文件的源程序*/
#include <stdio.h>
extern char string[];              /*定义外部字符一维数组*/
void upper(void)
{ int i;
  for (i=0;string[i]!=NULL;++i)
    if (string[i]>='a'&&string[i]<='z')
      string[i]-=32;
}
```

本程序包括两个源程序文件，可分别编制，然后连接生成为一个可执行文件(详见有关参考手册)，其运行结果为：

```
Please enter a string:
I__love__students.↙
I__LOVE__STUDENTS.
```

8.8　命令行参数

命令行参数就是指 main 函数的参数。虽然在前面见到的 main 函数后的括号是空的，但实际上 main 函数是可以带参数的。它的一般形式为：

数据类型 main(int argc,char * argv)

因 main 函数只作主调函数而不作被调函数，所以不能在程序内部取得实参值。要想取得实参值，我们从 DOS 操作系统命令行上赋予。它的一般形式为：

DOS 命令提示符 可执行文件名　实参　实参……；

注意：main 函数的两个形参和命令行中的参数在个数上不是一一对应的。形参只有两个，而实参可以不止两个，其中可执行文件名也算一个参数，argc 的值通常表示参数的个数，它一般由输入命令行时按输入实际参数的个数自动赋予。

例如，命令行为：C:\tc＞order xa yb 123；本例中“order”是一个参数，xa、yb、123 分别又是参数，所以 argc 的值为 4；argv 参数是字符串指针数组，其各元素值为命令行中各字符串的首地址，数组元素初值由系统自动赋予，本例赋值如图 8-18 所示。

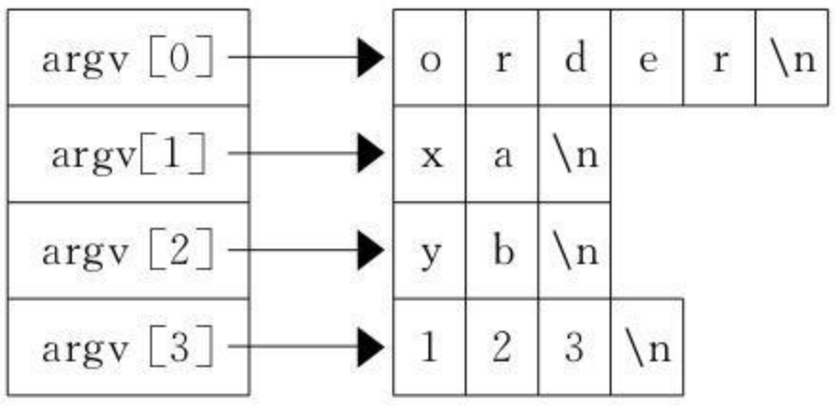


图 8-18　通过命令行赋予的字符串

我们通过一个程序来证实一下上面的结果，程序文件名为 order.c 经编译后为 order.exe。

【例 8.15】 命令行参数实例。

```
#include <stdio.h>
void main(int argc,char *argv[])
{ while (argc>1)
  { argv++;
    printf("%s\n", *argv);
    --argc;
  }
}
```

程序执行时，首先找到 ordey.exe 文件所在的文件夹，然后选择“开始/程序/附件/命令提示符”，打开“命令提示符”窗口，进入 order.exe 所在的文件夹，再在命令行中输入 order xa yb 123↙；则程序运行结果，如图 8-19 所示。

图 8-19　例 8.15 程序运行结果

8.9　例题精解

【例 8.16】 编写两个正整数的最大公约数的函数和它们最小公倍数的函数，并由主函数调用它们。

```
#include <stdio.h>
int gcd(int a,int b)
{ int r;
  do
  { r=a%b;a=b;b=r;
  }while(r!=0);
  return a;
}
int minp(int a,int b)
{
  int gcd(int,int),i;
  i=gcd(a,b);
  return a*b/i;
}
void main(void)
{
  int m,n,g,mil;
  printf("input m,n:");
  scanf("%d%d",&m,&n);
  g=gcd(m,n);
  mil=minp(m,n);
  printf("gcd=%d,minp=%d\n",g,mil);
}
```


图 8-20　例 8.16 程序运行结果

程序运行结果，如图 8-20 所示。

【解析】 先编出求最大公约数的函数 gcd，然后可应用于求最小公倍数的函数中

minp。求最大公约数用辗转相除法(a 作被除数,b 作除数,r 作余数,每循环一次 b 值给 a,r 值给 b,r 是 a 除以 b 的余数)。最小公倍数为要求的两数相乘,除以最大公约数(a * b/i)。

【例 8.17】 写出下列程序的运行结果。

```
#include<stdio.h>
int fun(int a)
{ int b=0;
  static int c=3;
  a=c++; b++;
  return a;
}
void main(void)
{
  int a=2, i,k=0;
  for(i=0; i<2; i++)
    k+=fun(a++);
  printf("%d", k);
}
```

图 8-21　例 8.17 程序运行结果

程序运行结果,如图 8-21 所示。

【解析】 函数 fun 中,c 为 static 变量,它只被赋初值一次,在 fun 的多次调用中 c 每次都在变化,且本次调用的结果就是下次调用的初值。第一次调用 c 为 3 赋值给 a,赋值后 c 自加为 4 值保存,第二次调用时 c 值为 4 赋值给 a,赋值后 c 自加为 5 值保存。所以 k=3+4 为 7。

【例 8.18】 利用函数调用打印如下图形。

```
        *
      * * *
    * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

```
#include <stdio.h>
void ptr(char c,int n)
{ if(n>0)
  { printf("%2c",c);
    ptr(c,n-1);
  }
}
void main(void)
{
  int i,n;
```

```
    printf("\ninput a number:\n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
      ptr(' ',50+n-i);
      ptr('*',2*i-1);
      printf("\n");
    }
    for(i=n-1;i>=1;i--)
    {
      ptr(' ',50+n-i);
      ptr('*',2*i-1);
      printf("\n");
    }
}
```

图 8-22 例 8.18 程序运行结果

程序运行结果,如图 8-22 所示。

【解析】 此程序定义一个 ptr 递归函数,实现打印菱形。在 main 函数中,第一个 for 循环实现打印菱形的上半部,第二个 for 循环实现打印菱形的下半部。在两个 for 循环中的两个 ptr 调用语句分别完成菱形的定位和打印个数。

【例 8.19】 把十进制数 m 转换成 n 进制数输出。

```
#include <stdio.h>
void f(int m,int n)
{
  if(m!=0)
  {
    f(m/n,n);
    switch(n)
      {
        case 2:
        case 8: printf("%d",m%n);break;
        case 16:
        m%n>=10? printf("%c",m%n-10+'a'):printf("%d",m%n);break;
        default: printf("No conside! \n");
      }
  }
}
void main(void)
{
  int m,n;
```

```
    printf("\ninput m,n:\n");
    scanf("%d%d",&m,&n);
    f(m,n);
}
```

程序运行结果,如图 8-23 所示。

a)将 255 转换成 2 进制数　　b)将 255 转换成 8 进制数　　c)将 255 转换成 16 进制数

图 8-23　例 8.19 程序运行结果

【解析】 编程思路:根据转换规则,可把 m 不断地用 n 去除,所得的余数即为 n 进制的数码。但由于先产生的余数是低位,后产生的是高位,因此必须先把后产生的余数输送出来。要达到这样的目的,可设计一个递归函数,在其中先是递归调用,直到达到边界条件时开始回返,而输出是在返回的过程中进行的,这样就可以实现后产生的先输出。

另外应注意:n 进制的数码可能超过 10,如 m%16 就是这样。这时应该对余数加以判断:如小于 10,则直接输出该数;如大于 10,则应把大于 10 的数用相应的字符输出。该字符为 m%n−10+'a'。

【例 8.20】 写一函数,使输入的一个字符串按逆序存放,在主函数中输入和输出字符串。

程序运行结果,如图 8-24 所示。

```
#inclnde <stdio.h>
#include <string.h>
void main(void)
{ void inverse(char b[]);
  char str[100];
  printf("Input string:");
  gets(str);
  inverse(str);
  printf("Inverse string: %s\n",str);
}
void inverse(char b[])
{ char temp;
  int i,j;
  j=strlen(b);
  for(i=0;i<j/2;i++)
  { temp=b[i];
    b[i]=b[j-i-1];
```

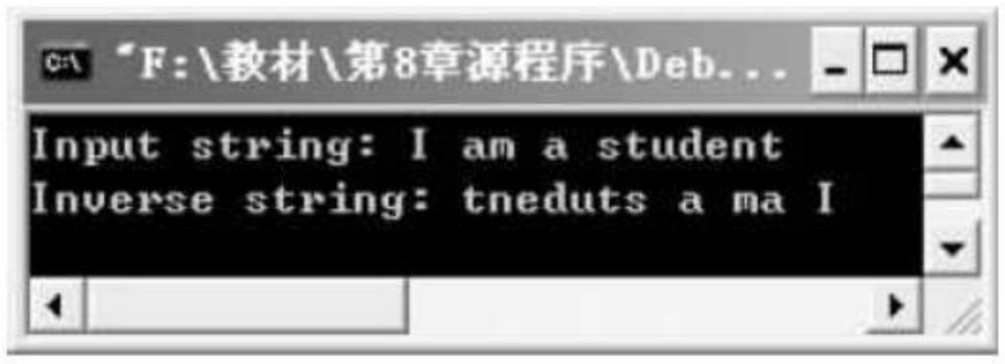


图 8-24　例 8.20 程序运行结果

```
        b[j-i-1]=temp;
    }
}
```

【解析】 函数 inverse 的形参定　char b[],表示字符数组作函数的参数。当函数调用时,实参是数组名,将数组 str 区域的首地址传给形参数组 b,使形参数组 b 与实参数组 str 是同一数组。在函数体执行时,对数组 b 的操作,实际就是对主调函数中实参数组 str 的操作。

8.10　本章小结

在结构化程序设计中,函数是将任务进行模块划分的基本单位。程序中除了主函数和系统函数外,用户可以自定义函数。C 语言中的用户自定义函数从 4 个方面定义:函数类型、函数名、形式参数表、函数体。

函数之间的数据联系是由函数间的数据传递建立的。数据传递方式分为:

(1) 传值方式:单向传递。形参、实参为不同的存贮单元,仅把主调函数的数据传到被调函数,子函数的任何改变,不会影响到主调函数。

(2) 传址方式:可看作双向传递。形参、实参指向同一存储区域,不仅能把主调函数的数据传到被调函数中,还可以把被调函数的值带回主调函数。

数组与函数,数组元素作为函数实参,实现的是值传递方式;数组名作为实参,实现的是地址传递方式。

C 语言不允许函数嵌套定义,但允许嵌套调用,调用时要注意对被调函数作适当说明。

递归调用是指一个函数可以直接或间接地调用自身,设计时要有递推调用的过程和回归过程,要推出递归公式和递归结束条件。

变量的作用域指变量在程序中的有效区域,分为局部变量和全局变量。局部变量的作用域为定义该变量的函数或复合语句内。全局变量的作用域为从定义到程序运行结束。

变量的存贮类型是指变量在内存中的存储方式,即变量的生存期,有静态存储和动态存储两种。

习　　题

一、选择题

1. C 语言程序是由_____构成。

A)主程序和子程序　　B)主函数和若干子函数

C)一个主函数和一个其他函数　　D)主函数和子程序

2. 以下说法中正确的是_____。

A)C 语言程序总是从第一个函数开始执行

B)在 C 语言程序中,要调用的函数必须在 main()函数中定义

C)C 语言程序总是从 main()函数开始执行

D)C 语言程序中的 main()函数必须放在程序的开始部分

3. 以下对 C 语言函数的有关描述中,正确的是_____。

A)调用函数时,只能把实参的值传送给形参,形参的值不能传送给实参

B)C 语言函数既可以嵌套定义又可以递归调用

C)函数必须有返回值,否则不能使用函数

D)C 程序中有调用关系的所有函数必须放在同一个源程序文件中

4. 对于 C 程序的函数,下列叙述中正确的是_____。

A)函数的定义不能嵌套,但函数调用可以嵌套

B)函数的定义可嵌套,但函数的调用不能嵌套

C)函数的定义和调用均不能嵌套

D)函数的定义和调用均可嵌套

5. 在 C 语言中,函数的隐含存储类别为_____。

A)auto　　B)static　　C)extern　　D)无存储类别

6. C 语言程序中,当函数调用时_____。

A)实参和形参各占一个独立的存储单元

B)实能和形参共用一个存储单元

C)可以由用户指定是否共用存储单元

D)计算机系统自动确定是否共用存储单元

7. 关于 return 语句,下列正确的说法是_____。

A)在主函数和其他函数中均要出现

B)必须在每个函数中出现

C)可以在同一个函数中出现多次

D)只在除主函数之外的函数中出现一次

8. 一个函数返回值的类型是由_____决定的。

A)return 语句中表达式的类型

B)在调用函数时临时指定的

C)定义函数时指定的函数类型

D)调用该函数的主调函数的类型

9. 在 C 语言的函数中,下列正确的说法是_____。

A)必须有形参　　B)形参必须是变量名

C)可以有也可以没有形参　　D)数组名不能作形参

10. 以下描述正确的是_____。

A)函数调用可以出现在执行语句或表达式中

B)函数调用不能作为一个函数的实参

C)函数调用可以作为一个函数的形参

D)以上都不正确

11. 在调用函数时,如果实参是简单变量,它与对应形参之间的数据传递方式是_____。

A)地址传递

B)单向值传递

C)传递方式由用户指定

D)由实参传给形参,再由形参传回实参

12. 当调用函数时,实参是一个数组名,则向函数传送的是_____。

A)数组的长度　　B)数组的首地址

C)数组每一个元素的地址　　D)数组中每个元素的值

13. 如果在一个函数的复合语句中定义了一个变量,则该变量_____。

A)只在该复合语句中有效,在该复合语句外无效

B)在该复合语句中任何位置都有效

C)在本程序的源文件范围内均有效

D)此定义方法错误,其变量为非法变量

14. 下列说法不正确的是_____。

A)主函数 main 中定义的变量在整个文件或程序中有效

B)不同函数中,可以使用相同名字的变量

C)形式参数是局部变量

D)在函数内部,可以在复合语句中定义变量,这些变量只在本复合语句中有效

15. 在一个源程序文件中定义的全局变量的有效范围是_____。

A)本源程序文件的全部范围

B)一个 C 程序的所有源程序文件

C)函数内全部范围

D)从定义位置开始到源程序文件结束

16. 以下叙述中不正确的是_____。

A)在不同的函数中可以使用相同名字的变量

B)函数中的形式参数是局部变量

C)在一个函数内定义的变量只在本函数范围内有效

D)在一个函数内的复合语句中定义的变量在本函数范围内有效

17. 如果要限制一个变量只能为本文件所使用,必须通过_____来实现。

A)外部变量说明　　B)局部自动变量

C)静态外部变量　　D)局部变量说明

18. 有如下函数调用语句

```
func(x1,x2+x3,(x4,x5));
```

该函数调用语句中,含有的实参个数是_____。

A)3　　B)4　　C)5　　D)有语法错误

二、填空题

1. 根据函数的作用范围函数分为_____、_____。

2. 函数的形式参数的作用域为_____,全局的外部变量和函数体定义的局部变量重名时,_____变量优先。

3. 若自定义函数要求返回一个值,则应在该函数中有一条_____语句,若自定义函数要求不返回一个值,则应在该函数说明时加一个类型说明符_____。

4. 设在主函数中有以下定义和函数调用语句,且 fun 函数为 void 类型;请写出 fun 函数的首部_____。(要求形参名为 b)

```
void main()
{ double s[10][20];
```

```
  int n;…;
  fun(s);…;
}
```

5. 完善下列程序。

```
#include <stdio.h>
int x;
void      (1)      ;                                    (1)_____
void main(void)
{ int x=1;
  if(x==1){int x=2;printf("%d",x++);}
  {      (2)      int x;printf("%d",++x);}              (2)_____
    fun();
  }
void fun(void)
{ printf("%d",x++);}
```

6. 下面函数的功能是：求 x 的 y 次方，请填空。

```
double fun(double x,int y)
{ int i;
  double z;
  for(i=1,z=x;i<y;i++)
  z=z*      (1)      ;                                 (1)_____
  return z;
}
```

7. 完善下列程序。

```
#include <stdio.h>
int imin(int,int);
void main(void)
{ int a,b,c;
  scanf("%d%d",      (1)      );                        (1)_____
  c=      (2)      (a,b);                               (2)_____
  printf("a=%d,b=%d,c=%d\n",a,b,c);
}
int imin(int x,int y)
{ int z;
  if (x<y)z=x;
  else z=y;      (3)      ;                             (3)_____
}
```

8. 分别计算并输出 1!,2!,3!,4! 和 5!。

```
#include <stdio.h>
```

```
int fac(int);
svoid main(void)
{ int i;
  for(i=1;i<=5;i++)
  printf("%d!=%d\n",i,_____(1)_____);
}
int fac(int n)
{ _____(2)_____f=1;
  f*=n;return(f);
}
```

(1)_____

(2)_____

三、应用题

1. 分析程序,写出运行结果。

```
#include <stdio.h>
unsigned fun(unsigned num)
{ unsigned k=1;
  do { k*=num%10;num/=10;
     }while(num);
  return(k);
}
void main(void)
{ unsigned n=26;
  printf("%d\n",fun(n));
}
```

2. 分析程序,写出运行结果。

```
#include <stdio.h>
int func(int,int);
void main(void)
{ int k=4,m=3,p;
  p=func(k,m);
  printf("%d",p);
  p=func(k,m);
  printf("%d\n",p);
}
int func(int a,int b)
{ static int m=0,i=2;
  i+=m+1;
  m=i+a+b;
  return (m);
}
```

3. 分析程序，写出运行结果。

```
#include <stdio.h>
void p1(void);
void p2(void);
static int a=5,b=3;
void main(void)
{ printf("a=%d,b=%d\n",a,b);
  p1();
  p2();
}
void p1(void)
{ printf("a*b=%d\n",a*b);
  return;
}
void p2(void)
{ printf("a+b=%d\n",a+b);
  return;
}
```

4. 分析程序，写出运行结果。

```
#include <stdio.h>
int fun(int x)
{ int p;
  if(x==0||x==1) return(3);
  p=x-fun(x-2);
  return p;
}
void main(void)
{ printf("%d\n",fun(9));
}
```

四、编写程序题

1. 写一个函数，连接两字符串。

2. 写一函数，输入一个四位数字，要求输出这四个数字字符，但每个数字间空一个空格。如输入 2005，输出为：2　0　0　5。

3. 写一函数，输入一个十六进制数，输出相应的十进制数。

4. 用递归求 n 阶勒让德多项式的值，递归公式为

$$P_n(x)=\begin{cases}1 & (n=0)\\ x & (n=1)\\ ((2^{n-1})*x*P_{n-1}(x)-(n-1)*P_{n-2}(x))/n & (n>1)\end{cases}$$

5. 定义函数 ch(c)，如字符 c 是小写字母就转成大写字母并通过 ch 返回，否则字符 c 不改变。

第9章 编译预处理

【教学提示】

ANSI C标准规定可以在C源程序中加入一些“预处理命令”(preprocessor directives),以改进程序设计环境,提高编程效率。编译预处理是C编译系统的一个组成部分,C语言源程序可以包含有多种预处理指令,在C编译系统对程序进行通常的编译前,先对程序中这些预处理指令进行预处理,然后将预处理的结果和源程序一起再进行一般的编译处理,从而得到目标代码(目标程序)。

【核心概念】

宏定义 文件包含 条件编译

编译预处理是C语言程序在编译过程之前所做的工作,它是由预处理程序负责完成。

C语言中所有的预处理命令都是以#开始,一行只能书定一个预处理命令。常用的编译预处理命令有三种:

(1)宏定义。

(2)文件包含。

(3)条件编译。

正确合理的使用编译预处理可以有效的提高程序的开发效率,改善程序的移植性。

9.1 宏定义

9.1.1 不带参数的宏定义

不带参数的宏定义,是指其后不带任何参数。

1. 格式

#define 标识符 字符串

2. 功能

编译之前,预处理程序将程序中该宏定义之后出现的所有标识符(宏名)用指定的字符串进行替换。在源程序通过编译之前,C的编译程序先调用C预处理程序对宏定义进行检查,每发现一个标识符,就用相应的字符串替换,只有在完成了这个过程之后,才将源程序交给编译系统。

例如,为了提高程序的可读性,将真和假的逻辑符号定义为TRUE与FALSE:

```
#define  TRUE  1
#define  FALSE  0
```

这样在编译时,每当在源程序中遇到TRUE和FALSE就自动用1和0代替。又如:

```
#define N 10
void main(void)
{ int a[N],i;
```

```
    for (i=0;i<N;i++)
      scanf("%d",&a[k]);
  ……
}
```

编译该程序之前，预处理程序首先将所有出现的 N 用 10 替换，这个过程叫宏展开。

3. 说明

(1)通常 #define 命令出现在文件开头，则在整个文件范围内有效，直到用 #undef 命令终止宏定义。

(2)宏定义是用宏名(标识符)代替一个字符串，只做简单的置换，不是赋值语句，不做语法检查。

(3)为了区别程序中其他的标识符，宏名的定义通常用大写字母。

(4)宏定义不需在行尾加分号。若加分号则带分号一起替换。

(5)双引号内的宏名不进行宏替换。

例如，执行：

```
#define PI 3.14159
printf("PI=%f",PI);
```

结果为

```
PI=3.14159
```

双引号中的 PI 不进行替换。

(6)在宏定义中，可引用已定义的宏名，可以层层置换。

例如：

```
#define   R   3.0
#define   PI  3.1415926
#define   L   2*PI*R
#define   S   PI*R*R
void main(void)
{
   printf("L=%f\nS=%f\n",L,S);
}
```

运行情况如下：

```
L=18.849556
S=28.274333
```

经过宏展开后，printf 函数中的输出项 L 被展开为 2 * 3.1415926 * 3.0，S 展开为 3.1415926 * 3.0 * 3.0，printf 函数调用语句展开为：

```
printf("L=%f\nS=%f\n", 2*3.1415926*3.0, 3.1415926*3.0*3.0);
```

使用宏可以有以下好处：

(1)在输入源程序时，可以节省很多操作。

(2)宏经定义之后，可以使用多次，因此，使用宏可以增强程序的易读性和可靠性。

(3)使用宏系统不需额外的开销，因为宏所代表的代码只在宏出现的地方展开，因此并

不会引起程序的跳转。

9.1.2 带参数的宏定义

1. 格式

#define 宏名(参数表) 字符串

字符串中包含在括号中所指定的参数。

2. 功能

预处理程序将程序中出现的所有带实参的宏名,展开成由实参组成的表达式。

例如:

```
#define S(a,b) a * b
...
area=S(3,2);
```

定义矩形面积 S,a 和 b 是边长。在程序中用了 S(3,2),用 3、2 分别代替宏定义中的形式参数 a、b,即用 3 * 2 代替 S(3,2),因此赋值语句展开为:

```
area=3 * 2;
```

3. 说明

(1)与不带参数的宏定义相似,宏名和参数都只做简单的替换。例如:

```
#define A(s) s/3+1
```

则凡遇到 A(s)就将它替换为 s/3+1,若程序中有 A(9),就替换为 9/3+1 即为 4,若程序中有 A(3+6)则替换为 3+6/3+1,结果为 6,而不是 4。当参数为表达式时,最好在宏定义时用括号将参数括起来。

```
#define A(s) (s)/3+1
```

再遇到 A(3+6)就替换为(3+6)/3+1,即为 4。

(2)宏名与带参数括号间不可有空格。例如:

```
#define a (s) s/3+1
```

则在程序中遇到标识符 a 替换为(s) s/3+1,遇到 a(s)不替换。

4. 带参数宏定义与函数的比较

有些读者容易把带参数的宏和函数混淆。的确,它们之间有一定类似之处,在引用函数时也是在函数名后的括弧内写实参,也要求实参与形参的数目相等。但是带参的宏定义与函数是不同的。主要有:

(1)函数调用时,先求出实参表达式的值,然后代入形参。而使用带参的宏只是进行简单的字符替换。

(2)函数调用是在程序运行时处理的,分配临时的内存单元。而宏展开则是在编译时进行的,在展开时并不分配内存单元,不进行值的传递处理,也没有“返回值”的概念。

(3)对函数中的实参和形参都要定义类型,两者的类型要求一致,如不一致,应进行类型转换。而宏不存在类型问题,宏名无类型,它的参数也无类型,只是一个符号代表,展开时代入指定的字符即可。宏定义时,字符串可以是任何类型的数据。例如:

```
#define STRING welcome to you(字符串)
#define A 3.6(数值)
```

STRING 和 A 只是标识符，不是变量，所以也不需要定义类型。又如：

```
#define  A(s)  s/3+1
```

假设程序中的 s 均等于 9，当完成了宏替换后，源程序中所有的 A(s)都被替为 9/3+1，进入程序编译时，在源程序中已没有了 A 这个宏名，也没有了 s 这个参数，所以根本不用定义参数的类型。

(4)使用宏次数多时，宏展开后源程序变长，因为每展开一次都使程序增长，而函数调用不会使源程序变长。

(5)宏替换不占运行时间，只占编译时间。而函数调用则占运行时间(分配单元、保留现场、值传递、返回)。

总而言之，宏代换的操作就相当于我们在 C 语言源程序编译前，做了若干次的寻找与替换的编辑操作，是一种文本编辑的操作。

一般用宏来代表简短的表达式比较合适，有些问题用宏和函数都可以。请看下面的例子。

```
#define  MAX(x,y)  (x)>(y)? (x):(y)
void main(void)
{ int a,b,c,d,t;
  t=MAX(a+b,c+d);
}
```

赋值语句展开后为：

```
t=(a+b)>(c+d)? (a+b):(c+d);
```

注意：MAX 不是函数，这里只有一个 main 函数，在 main 函数中就能求出 t 的值。这个问题也可用函数来求。

```
#include <stdio.h>
int imax(int x,int y)
{ return(x>y? x:y);}
void main(void)
{ int a,b,c,d,t;
  t=imax(a+b,c+d);
}
```

imax 是函数，在 main 函数中调用 imax 函数才能求出 t 的值。

9.2　文件包含

“文件包含”是指在一个源文件中可以包含另一个源文件，即把另一个源文件插入到该文件中。

1. 格式

#include<文件名>或 #include“文件名”

2. 功能

使文件包含命令中所指定的文件为当前文件所包含,即将文件包含命令所指定的文件内容全部插入当前的文件中。例如:

调用系统库函数中的字符串处理函数,需在程序的开始使用:

#include<string.h>

表明将 string.h 文件的内容,嵌入当前文件中。

3. 说明

(1)被包含的文件为源文件,内容主要是宏定义、结构体的定义及全局变量的定义等,相当于把被包含文件复制到当前文件中,那么其必然是源文件,而不是目标文件,与连接(link)是不同的。

所以可看出,头文件实质上也就是一个没有函数,只有定义的C语言源文件。我们将它们另定义一类称为头文件,采用".h"作为文件后缀名,这样可更好地区分它与一般的C语言源程序。

(2)一条文件包含命令只能指定一个被包含文件。

(3)正如宏定义可嵌套,文件包含也如此,如 prog.c 中包含文件 file1.c,在 file1.c 中需包含文件 file2.c,可以在 prog.c 中使用两个#include 命令,分别包含 file1.c 和 file2.c,而且 file2.c 应当写在 file1.c 的前面。即:

```
#include<file2.c>
#include<file1.c>
```

(4)在文件包含命令中,文件名用双引号与用尖括号括起来的区别。

#include <文件名>

预处理程序在标准目录下查找指定的文件。

#include "文件名"

预处理程序首先在引用被包含文件的源文件所在的目录中寻找指定的文件,如没找到,再按系统指定的标准目录查找。

9.3 条件编译

一般情况下,源程序中所有的行都参加编译。但是有时希望对其中一部分内容只在满足一定条件才进行编译,也就是对一部分内容指定编译的条件,这就是"条件编译"。

条件编译命令有以下几种形式:

1. 第一种

(1)格式

```
#ifdef  标识符
    程序段 1
#else
    程序段 2
#endif
```

(2)功能

如果标识符在前面进行了宏定义，则编译程序段 1，否则编译程序段 2。对于提高 C 语言程序的通用性很有好处，另外使用条件编译可使程序在调试过程中，编译执行一些调试命令语句，而当正常运行后，跳过调试命令。例如：

```
#ifdef  DEBUG
   printf("%d,%f",aa,bb);
#endif
```

在程序调试过程中，用 printf("%d,%f",aa,bb)命令将变量当前值按格式输出到显示器，跟踪观察变量 aa，bb 的变化，而正常运行时不要显示，则可用上述命令来实现。

2. 第 2 种

(1)格式

```
#ifndef  标识符
   程序段 1
#else
   程序段 2
#endif
```

(2)功能

如果未定义标识符，编译执行程序段 1，若已定义了标识符，就编译执行程序段 2。

3. 第 3 种

(1)格式

```
#if  表达式
   程序段 1
#else
   程序段 2
#endif
```

(2)功能

如果表达式为真，则编译执行程序段 1，否则编译执行程序段 2。可以事先给定一定条件，使程序在不同的条件下执行不同的功能。

注意：条件语句与条件编译的差别：条件语句是对整个源程序进行编译，生成的目标代码程序很长；而条件编译，则根据条件只编译其中的程序段 1 或程序段 2，生成的目标程序较短。因而采用条件编译，可以减少目标文件的长度。

9.4　例题精解

【例 9.1】　分析以下程序的结果。

```
#include <stdio.h>
#define  PI  3.1415926
#define  S(r)  PI * r * r
void main(void)
```

```
{ float a,area;
  a=3.6;
  area=S(a);
  printf("r=%f\narea=%f\n",a,area);
}
```

图 9-1　例 9.1 程序运行结果

程序运行结果,如图 9-1 所示。

【解析】 此程序定义了两个宏名。一个为不带参数的宏定义,另一个为带参数的宏定义,PI 在程序中直接引用,而赋值语句 area=S(a);经宏展开后为 area=3.1415926 * a * a;再计算,将表达式的值赋给变量 area。

【例 9.2】 求 1 到 10 平方。

方法一:使用函数。

```
#include <stdio.h>
void main(void)
{ int i=0;
  while(i<=10)
     printf("%d,",fun(i++));
}
fun(int k)
{ return(k * k);
}
```

程序运行结果,如图 9-2 方法一所示。

方法二:使用宏。

```
#include <stdio.h>
#define FUN(a) a * a
void main(void)
{ int k=1;
  while(k<=10)
     printf("%d",FUN(k++));
}
```

程序运行结果。如图 9-2 方法二所示。

方法一

方法二

图 9-2　例 9.2 程序运行结果

【解析】 程序中使用了两种不同的方法求解 1 到 10 的平方。两种方法比较可以看出，方法二效率更高。

【例 9.3】 输入一行字母字符，根据需要设置条件编译，使之能将字母全改为大写输出，或全改为小写字母输出。

```
#include <stdio.h>
#define  LETTER  1
void main(void)
{ char str[20]="C Language",c;
  int i;
  i=0;
  while((c=str[i])!='\0')
  { i++;
  #if LETTER
    if (c>='a'&& c<='z')
      c=c-32;
  #else
    if (c>='A'&& c<='Z')
      c=c+32;
  #endif
    printf("%c",c);
    }
}
```


图 9-3　例 9.3 程序运行结果

程序运行结果，如图 9-3 所示。

【解析】 此程序通过条件编译的方法，将数组中存放的字符串小写改为大写，或大写改为小写。根据给定的宏定义，此程序是把字符串中的所有小写改为大写。

思考：如果将程序第一行改为

```
#define  LETTER  0
```

运行结果是什么？

9.5　本章小结

预处理是 C 语言特有的功能，它优化了程序设计环境，简化了程序开发过程。是在对源程序正式编译前由预处理程序完成的。

(1)宏定义：用一个标识符来表示一个字符串，宏调用时用字符串代换宏名，宏定义可带有参数，代换时是以实参代换形参，而不是“值传送”。

(2)文件包含：把多个源文件连接成一个源文件进行编译，结果将生成一个目标文件。

(3)条件编译：只编译程序中满足条件的程序段，从而生成的目标代码较短，减少了内存开销，方便了程序的调试和移植。

习　　题

一、单项选择题

1. 下列说法正确的是_____。

A)C 程序必须在开头用预处理命令＃include

B)预处理命令必须位于 C 源程序的首部

C)在 C 语言中,预处理命令都以“＃”开头

D)C 语言的预处理命令只能实现宏定义和条件编译的功能

2. 有以下宏定义

```
#define N 2
#define Y(n) ((N+1) * n)
```

则表达式 z=2 *(N+Y(5))的值为_____。

A)34　　B)70.0　　C)已无定值　　D)表达式有误

3. 程序输出结果为_____。

```
#include <stdio.h>
#define MOD(x,y) x%y
void main(void)
{int z,a=15,b=100;
 z=MOD(b,a);
 printf("%d\n",z++);
}
```

A)11　　B)10　　C)0　　D) 宏定义不合法

4. 下列预处理命令正确的是_____。

A)＃include <stdio.h>;　　B)define M

C)＃define M 3　　D)define M 3+5;

二、填空题

1. 设有以下宏定义

```
#define  F(N)  3 * N
```

则表达式 F(2+3)的值是_____。

2. 设有下列宏定义

```
#define  A  3+3
#define  B  A * A
```

则表达式 B/B 的值为_____。

3. 设有宏定义

```
#define  ABC(A,B,C)(A)? (B) : C
```

则表达式 ABC(ABC(1,2,3),ABC(3,2,1),ABC(2,3,1))的运算结果是_____。

4. 下列带参宏的功能是交换两个参数值,请填写所缺少的语句。

```
#define  H(x,y){x=x+y;y=x-y;_____;}
```

三、应用题

1. 阅读下列程序，写出运算结果。

```
#include <stdio.h>
#define T 6.5
#define S(x) T*x*x
void main(void)
{ int a=2,b=3;
  printf("%4.1f\n",S(a+b));
}
```

2. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
#define PR(ab) printf("%d",ab)
void main(void)
{ int j,a[]={1,3,5,7,9,11,13,15},*p=a+5;
  for(j=3;j;j--)
   switch(j)
   { case 1:
     Case 2:PR(*P++);break;
     case 3:PR(*(--P));
   }
}
```

3. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
#define AB(A) A? 1:0
void main(void)
{ char s[]={"1 2 3 4 5 6 7 8 9 0"},*p=s;
  int i=0;
  do
  { printf("%c",*(p+i));
    #if AB(1)
      i=5;
    #else
      i++;
    #endif
  } while(i<10);
}
```

四、编写程序题

1. 定义一个带参数的宏，使两个参数的值互换，并写出程序，输入两个数作为使用宏时的参数，输出已交换后的两个值。

2. 三角形的面积公式为

$$area=\sqrt{s(s-a)(s-b)(s-c)}$$

其中 $s=\frac{1}{2}(a+b+c)$，a、b、c 为三角形的三边。定义两个带参数的宏，一个用来求 s，另一个宏用来求 area。写程序，在程序中用带实参的宏名来求面积 area。

3. 分别用函数和带参数的宏，从 3 个数中找出最大数。

4. 采用条件编译，使给定的字符串按小写字母输出或按大写字母输出。

5. 用条件编译方法实现以下功能：

输入一行电报文字，可以任选两种输出：一为原文输出；一为将字母变成其下一字母（如'a'变成'b','b'变成'c'…'z'变成'a'。其他字符不变）。用 #define 命令来控制是否要译成密码。例如：

#define CHANGE　1

则输出密码。若//#define CHANGE　0

则不译成密码，按原码输出。

第 10 章　结构体与共用体

【教学提示】

在程序设计的应用中常常会遇到需要表示同一对象的不同特征，每个特征用不同的数据类型表示，而反映对象不同特征的数据要作为一个整体对象。如表示一个学生基本信息的数据有：姓名(char)、性别(char)、年龄(int)、成绩(float)等，显然对这些不同的数据类型无法用一个数组来存放。C 语言根据需要，可以将不同的基本类型通过特定的方式构造出用户所需要的类型。

【核心概念】

结构体类型　共用体类型　成员项　枚举　用户定义类型

10.1　结构体

结构体是一种构造类型，它可以将描述同一事物不同特征的数据构造成一个整体数据类型。如学生的基本情况：

字段	学号	姓名	性别	籍贯	分数
类型	long	char	char	char	float
实例	20057345	王晓	女	合肥	80.5

显然，在这组数据中，每项数据表示的是同一个学生基本信息的不同内容，学号是长整型，姓名、性别、籍贯是字符型数组，一门考试的成绩是浮点型。这些数据之间存着逻辑关系，反映了一个学生对象不同特征的数据。C 语言提供了由用户按照实际应用构造类型的方式。要使用这类具体的数据必须先定义结构体类型，其中各个数据成为结构体类型的成员项，每个成员项都有其相应的类型。

10.1.1　结构体类型的定义

结构体类型的定义是根据实际相关数据的具体情况，由用户定义的一种类型，定义结构体类型的一般形式为：

```
struct   结构体名
{   type   成员 1;
    type   成员 2;
        …
        …
    type   成员 n;
};
```

定义结构体类型的过程实际上说明了如下两点：

1. 结构体类型的名字；

2. 在定义的结构体类型中，各个成员项的名字和类型(由 type 说明)。

例如：

```
struct   Student
{
   long   iNum;
   char   chpName[20];
   int   iAge ;
   char   chAddress[30];
   float   fScore;
};
```

结构体类型定义后，类型名是 struct Student，该类型中包含如下成员项：

名字	iNum	chpName[20]	iAge	chAddress[30]	fScore
类型	long	char	int	char	float

10.1.2 结构体变量的说明

定义结构体，完成了结构体类型以及成员项的组成和类型的构造，提供了结构体的数据的组织方法，并没有提供具体的数据内容。而使用结构体，必须由定义的结构体类型说明结构体变量后，才开辟相应的内存空间，才能使用变量所具有的成员项内容。

说明结构体变量的方式有三种：

1. 先定义结构体类型，然后说明结构体变量

```
struct   Student
{
   long   iNum;
   char   chpName[20];
   int   iAge;
   char   chAddress[30];
   float   fScore;
};                                          /*定义结构体类型*/
struct   Student   fang , wang , zhang ;  /*说明结构体变量*/
```

定义结构体类型后，说明了三个变量，名字分别为 fang、wang、zhang，每个变量都具有结构体类型中的各个成员项。

2. 定义结构体类型的同时说明结构体变量

```
struct   Student
{
   long   iNum;
   char   chpName[20];
```

```
    int  iAge;
    char  chAddress[30];
    float  fScore;
} fang , wang , zhang;
```

定义结构体类型的同时，说明结构体变量 fang、wang、zhang。

3. 直接通过无名结构体说明变量

```
struct
{
    long  iNum;
    char  chpName[20];
    int  iAge;
    char  strAddress[30];
    float  fScore;
} fang, wang , zhang;
```

结构体的成员项可以是各种类型的变量，包括基本类型、指针类型、构造类型，如已定义好的结构体变量。例如：

```
struct  data
{  int  year;
   int  moon;
   int  day;
};
struct  Student
{
   long  iNum;
   char  chpName[20];
   int  iAge;
   char  strAddress[30];
   float  fScore;
   struct data birthday; /* 结构体变量 struct  data  birthday 作为 struct Student 的成员项 */
};
```

10.1.3　结构体变量的初始化

结构体变量与简单变量以及数组一样，可以在说明的时候对结构体变量赋初值，也就是给结构体变量的各个成员项赋初值。如：

```
struct  Student
{
   long  iNum;
   char  chpName[20];
   int  iAge;
```

```
    char   strSex[3];
    char   strAddress[30];
    float   fScore;
} zhang = {20021201 ,"张三" , 25,"男" , "合肥",90.5};
```

赋初值的方式是在说明结构体变量时,将各成员的常量列表按照成员项的类型及顺序在{ }中列出。

10.1.4 结构体变量的引用

结构体变量的引用必须以成员项作为引用的基本单位,也就是说,只能引用结构体变量的成员项而不能引用整个变量(函数参数以及函数返回值除外)。

引用方式:

结构体变量名 . 成员项名

【例 10.1】 有如下程序片段。

```
struct   data
{   int   year;
    int   moon;
    int   day;
};
struct   Student
{
   long   iNum;
   char   chpName[20];
   int   iAge;
   char   strAddress[30];
   float   fScore;
   struct   data   birthday;
}wang;
wang.iNum=20023456L;                          /*引用 wang 的 iNum 成员项*/
wang.birthday.year=1983;
   /*引用结构体变量 wang 中的结构体变量成员项 birthday 的 year 成员项*/
```

说明:

(1)结构体变量的各个成员项,具有同类型变量的一切特征,因此,可以在程序中把结构体变量的成员项当作对应类型的变量或数组元素使用。

```
如:wang.iAge++;                              /*参加运算*/
    strcpy(wang.chpName, "王五");            /*字符函数参数*/
```

(2)一般不能将结构体变量作为整体参加表达式的运算或输入输出。

```
    wang + 2/4;                               /*错误的应用*/
```

(3)“.”运算符是取结构体变量成员项的运算符,具有最高的运算级别。

10.1.5　结构体数组

当相同类型的变量有多个，特别是这些变量是具有联系的，如一个班级 45 个同学的基本情况用结构体变量表示，最好的方法是将 45 个结构体变量构造成数组。数组是同类型变量的有序集合。结构体变量也可以构造成数组，称为结构体数组。每个结构体数组元素都是一个结构体变量，都含有结构体的各个成员项。每个数组的元素在内存中的地址是按照数组元素下标的顺序连续的。

1. 结构体数组的说明

与结构体变量说明类似，可以通过三种形式，说明结构体数组。基本的说明方式如下：

struct　结构体名　结构体数组名[整型常量表达式]；

例如：

```
struct Student
{
   long   iNum;
   char   chpName[20];
   int   iAge;
   char   strAddress[30];
   float   fScore;
}stud[3];
```

当说明 stud 是一个具有三个元素的结构体数组后，其具有如下特征：

(1)stud[0]，stud[1]，stud[2]是三个结构体数组的元素，每个元素都含有结构体 Student 类型的各个成员项。

(2)结构体数组名 stud 代表结构体数组在内存中的首地址。

2. 结构体数组的初始化

结构体数组在说明时，可以对数组的部分或全部元素赋初值，即对数组元素的各个成员项初始化。如：

```
struct   Student
{
   long   iNum;
   char   chpName[20];
   int   iScore;
};
struct   Student stu[3] ={
                          {99010101,"张晓宇" ,75},
                          {99010102,"赵涛" ,78},
                          {99010103,"刘丽" , 94}
                         } ;
```

其内存基本分配图，如图 10-1 所示。

数组名 stu 是数组在内存中的首地址。

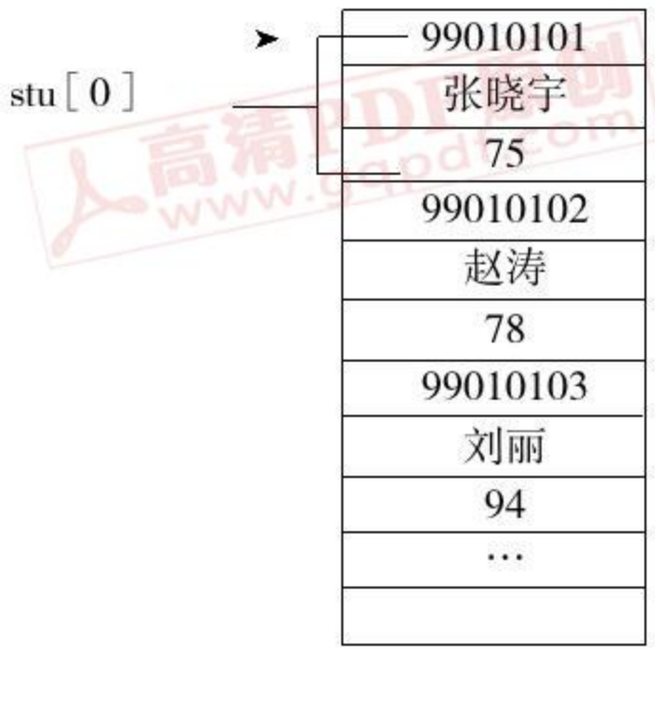


图 10-1 内存基本分配图

【例 10.2】 已知 20 人(人数可以通过宏定义改变)参加选举,共三个候选人,候选人的基本信息定义在结构体类型当中,包括候选人的标识号 iID(整型)、候选人的姓名 chpName(字符型数组)、得票数 iCount(整型)。选举时投票即输入三个人的标识号(iID 成员项,分别取 1、2、3)。

```
#include <stdio.h>
#define  NUMBER  20
struct  Student
{
  int  iID;
  char  chpName[20];
  int  iCount;
} lead[3]={ 1 ,"Zhang San" , 0 , 2 ,"Li  Si" , 0 , 3 ,"Wang Wu" , 0 };
/*定义结构体类型,说明结构体数组并初始化*/
void main (void)
{
  int  i ,j ,num;
  for (i=0 ; i<NUMBER ; i++ )
  {
    scanf ("%d" ,&num );                    /*循环输入候选人的标识号*/
  switch (num)
  {
    case 1 : lead[0]. iCount ++; break;
    case 2 : lead[1]. iCount ++; break;
    case 3: lead[2]. iCount ++; break;
    default : printf ("Error selection\n" );    /*输入错误,打印提示*/
  }
}
for (i=0 ; i<3 ; i++)
```

```
        printf ("\n%-10s:%-d" , lead[i]. chpName , lead[i]. iCount );
            /*输出候选人的姓名及的票数*/
}
```

程序运行结果,如图 10-2 所示。

图 10-2　例 10.2 程序运行结果

10.1.6　结构体指针

指针是特殊的数据类型,指针可以指向内存中的对象,然后通过指针访问所指向的内存对象,结构体变量或数组也是内存中的对象,因此,可以通过指针进行访问。指向结构体变量的指针称为结构体指针,可以指向结构体变量的首地址,其逻辑上加 1,等于加一个结构体类型所占的字节数,即加一个结构体的长度。结构体指针必须先说明,然后指向同类型的对象再通过指针引用所指对象的各个成员项。

1. 结构体指针的说明

格式:

struct　结构体名　*结构体指针名;

例如:

```
struct  stud_score { long  num;
                     char  name[20];
                     float  score;
                   };
struct  stud_score   *spoint;
```

spoint 被说明成一个指向 struct stud_score 结构体类型的指针,可以存放 struct stud_score 类型结构体变量或数组的地址。

2. 通过指针引用结构体对象

说明结构体指针后,必须让结构体指针指向同类型的结构体变量或数组;然后才能通过指针引用所指对象的成员项。结构体指针主要用于对结构体数组访问。

```
struct  stud_score {
                     long  iNum ;
                     char  chpName[20];
                     int  iScore;
                   }wang,student[45];
```

```
/* 定义结构体类型,说明结构体变量 wang,结构体数组 student */
struct  stud_score  * point1, * point2;     /* 说明指向结构体的指针,point1,point2 */
point1=&wang;                               /* point1 指向结构体变量 wang */
point2=student;                             /* point2 指向结构体数组 sttudent */
```

通过结构体指针访问所指向变量或数组元素的成员项有以下两种方式：

方式 1：**(*结构体指针名).成员项名**

例如：(*point1).iNum，即 wang.iNum。

> 注意：()不能省略，原因是“.”优先级比 * 高，如果没有括号()，表达式的含义将变为 *(point1.num)，即求 point1.num 作为地址所指向的内容，显然与语法不符。

方式 2：**结构体指针名->成员项名**

例如：point2->iNum; 即 student[0].iNum。

其中“->”运算符，表示取结构体指针所指向的结构体变量或结构体数组元素的成员项。

【例 10.3】 通过结构体指针输出结构体数组的各成员。

```
#include <stdio.h>
void main (void)
{ int  i;
  struct  sample {
                  unsigned  num;
                  char  name[20];
                  char  addr[30];
                 }w[]={
                       {20101,"张三","Shanghai"},
                       {20102,"李四","Bejing"},
                       {20103,"刘祥","Hefei"}
                      } ;  /* 定义结构体类型,并说明结构体数组 w */
  struct sample *ps ;
  ps=w ;                          /* 说明结构体指针 ps,指向结构体数组 w */
  for ( i=0; i<3; i++)
  {
    printf("%-9u%-14s%-15s\n",ps->num,ps->name,ps->addr);
    ps++;                         /* 指针加 1 指向下一个结构体数组元素 */
  }
}
```

程序运行结果，如图 10-3 所示。

图 10－3　例 10.3 程序运行结果

10.1.7　结构体与函数

结构体变量或结构体指针可以作为函数的参数或函数的返回值，这时结构体变量可以整体引用。

1. 结构体与函数参数

当函数的形参需要使用一个结构体变量时，一般有两种处理办法：

(1)传递一个结构体指针

将函数的形参定义成一个指向结构体的指针，调用时实参传递一个结构体变量的地址。由于在函数调用的参数传递过程中，只是传递一个地址，因此，系统的开销较小，效率较高。

【例 10.4】 通过函数，输出结构体变量的各个成员项。

```
#include  <stdio.h>
void   mprintp (struct student  * sp);      /* 函数的原型声明 */
struct   Student {
                  long   num;
                  char   name[20];
                  float   score;
                 };
void   main(void)
{
   struct Student s1={20053401,"王海",96.6};/* 说明结构体变量 s1 并赋初值 */
   mprintp (&s1) ;          /* 调用函数 mprintp,实参是结构体变量 s1 的地址 */
   printf("\n%6.1f",s1.score);
                            /* 函数通过指针可以修改实参结构体变量成员项的值 */
}
void   mprintp( struct Student  * sp)      /* 定义函数,参数是指向结构体的指针 */
{
   printf("%ld\n%s\n%8.1f\n",sp->num, sp->name, sp->score);
   sp->score=93.5;                  /* 通过指针修改 s1.score 的值 */
}
```

程序运行结果，如图 10－4 所示。

显然，程序中函数调用，只需完成将 &s1 传递给 sp，即传递一个地址，具有较高的效率。如果在函数中改变指针指向的某个成员项的值，结果会影响到实参的值。s1.score 原值是 96.6，通过调用函数中对 sp->score 的赋值，将其修改成 93.5。

(2)传递一个结构体变量

将函数的形参定义成一个结构体的变量,调用时实参传递一个结构体变量。在函数调用的参数传递过程中,结构体变量的各个成员项都要一一传递,传递的内容显然比传递一个地址要多,因此,系统的开销较大,效率低。

图 10-4 例 10.4 程序运行结果

【例 10.5】 下面程序是函数参数传递一个结构体变量的实例。

```
#include <stdio.h>
void  mprints(struct  student);              /*函数的原型声明*/
struct  Student{
                long  num;
                char  name[20];
                float  score;
               };
void  main(void)
{
  struct  Student  s1={20021345,"刘丽",90.0}; /*说明结构体变量 s1 并赋初值*/
  mprints(s1);                /*调用函数 mprints,实参是结构体变量 s1*/
}
void  mprints(struct  Student  sv)
{
  printf("%ld\n%s\n%6.1f\n", sv.num, sv.name, sv.score);
}
```

程序运行结果,如图 10-5 所示。

mprints( )是一个函数,函数的形参是一个结构体变量,在调用函数时,实参 s1 的各个成员项将传递给形参 sv 的各个成员项。

图 10-5 例 10.5 程序运行结果

2. 结构体与函数返回值

(1)函数的返回值可以是结构体变量,也可以是指向结构体变量的指针。当函数的返回值是一个结构体变量时,称该函数为一个结构体型函数,其一般形式为:

struct 结构体名 函数名(形参表)

{/*函数内容*/…}

【例 10.6】 结构体包含了一个学生的学号、姓名、成绩等成员项。结构体数组中包含了一个班 20 人的基本信息,通过函数输入结构体数组每个元素各个成员项的值。计算出成绩的平均值,输出成绩小于平均值的学号、姓名、分数。

```
#include<stdio.h>
#include<stdlib.h>
#define  NUM  3
```

```
struct  stu {
            long    iNum;               /* 学号 */
            char    chpName[20];        /* 姓名 */
            double  score;              /* 一门课的成绩 */
           };                           /* 定义结构体类型 */
struct stu  inp(void);                  /* 函数的原型声明 */
void  main (void)
{
  int i;
  double ave=0.0;
  struct  stu  ws[NUM];                 /* 说明结构体数组 */
  for(i=0; i<NUM;i++)
  {
    ws[i]= inp( );                      /* 循环调用 inp 函数,输入 ws 的各元素 */
    ave+=ws[i].score;                   /* 分数的累加和 */
  }
   ave/=NUM;                            /* 平均分 */
   printf("ave=%f\n",ave);
   for(i=0; i<NUM;i++)
      if(ws[i].score<ave)
      printf("学号:%ld 姓名:%s 成绩:%8.2f\n",ws[i].lNum,ws[i].chpName,
    ws[ i].score );                     /* 低于平均成绩,输出结果 */
}
struct  stu  inp(void)                  /* 结构体输入函数,类型为结构体,即返回一个结构体变量 */
{
  char str[20];
  struct  stu  stemp ;
  printf("\nInput Num:");
  gets(str);                            /* 以字符串形式输入长整型,存放到 str 数组中。 */
  stemp.iNum=atol(str);                 /* 通过函数 atol( )将 str 转换成 long 型赋给 stemp.iNum */
  printf("\nInput Name:");
  gets(stemp.chpName);                  /* 输入姓名 */
  printf("\nInput Score:");
  gets(str);                            /* 以字符串形式输入 double 型,存放到 str 数组中。 */
  stemp.score=atof(str);                /* 通过函数 atof( )将 str 转换成 double 型赋给 stemp.score */
  return (stemp);                       /* 完成输入,返回结构体变量 */
}
```

程序运行结果,如图 10-6 所示。

图 10-6 例 10.6 程序运行结果

在例 10.6 中函数 inp( )的类型是结构体变量,每次调用函数可输入一个结构体变量的各个成员项,并将结构体 stemp 变量返回,赋值给结构体数组元素 ws[i]。

(2)当函数的返回值为一个结构体指针时,该函数为一个结构体指针型函数。函数的类型说明方式为:

struct 结构体名 * 函数名(形参表)

{ /* 函数内容 */…}

【例 10.7】 结构体指针型函数举例。结构体包含了一个学生的学号、姓名、分数。结构体数组中包含了一个班(假定 3 人)的基本信息,输入一学号通过函数在结构体数组中查找,找到后函数返回数组元素的地址。

```
#include<stdio.h>
#define  NUM  3
struct  stu{
            long  num;
            char  name[20];
            float  score;
           };
struct  stu  *findp(struct stu sp[]);          /*函数原型声明*/
void  main(void)
{
  struct  stu  sa[NUM]={
                        20201011,"Fang", 89,
                        20201021,"Zhang", 99,
                        20201031,"Wang", 78};
  int  i;
    struct stu *sap;
    sap=findp(sa);          /*调用函数返回查找结果*/
    printf("number: %ld\n", sap->num);
    printf("name: %s\n", sap->name);
    printf("score: %f\n", sap->score);
}
```

```
struct stu    * findp (struct stu sp[])
{
   int   i;
   struct   stu    * p;
   p=sp;
   for (i=0; i<NUM; i++,p++)
      if (p->num ==200201011)
   return (p);
}
```

图 10-7　例 10.7 程序运行结果

程序运行结果，如图 10-7 所示。

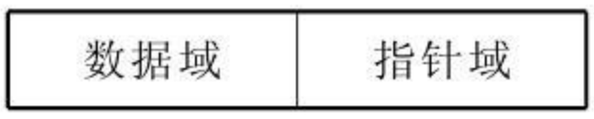
在例 10.7 中，函数的类型定义为指向结构体的指针。当代到对应的学号时，将其地址返回到主调函数输出。

10.1.8　链表

1. 链表的概念

数组是一种静态的存储方式。所谓静态存储方式表现在数组使用之前，数组元素个数已经说明好了，而且所占用的内存空间是连续的。因此，数组在处理一些动态数据(无法确切知道数据的项数，并有可能增减)时存在一定的局限。要求数组必须说明的足够大，如果数据项较少会造成内存使用的浪费，如果数据项多于说明，有可能造成程序错误。

链表是一种动态分配内存的数据组织方式。该方式允许用户根据需要随时增减数据项，而且，数据项在内存中不必连续。那么如何保证程序对链表中各项数据的访问呢？在链表数据中，任意一个数据项都包含如下内容：



数据域存放该项数据的内容，指针域存放下一数据项地址。每个数据项称为一个结点，每个结点不仅存放了数据，而且指明了下一项数据到哪里去取(指针域)。由此，通过指针将数据项链接在一起，如图 10-8 所示。

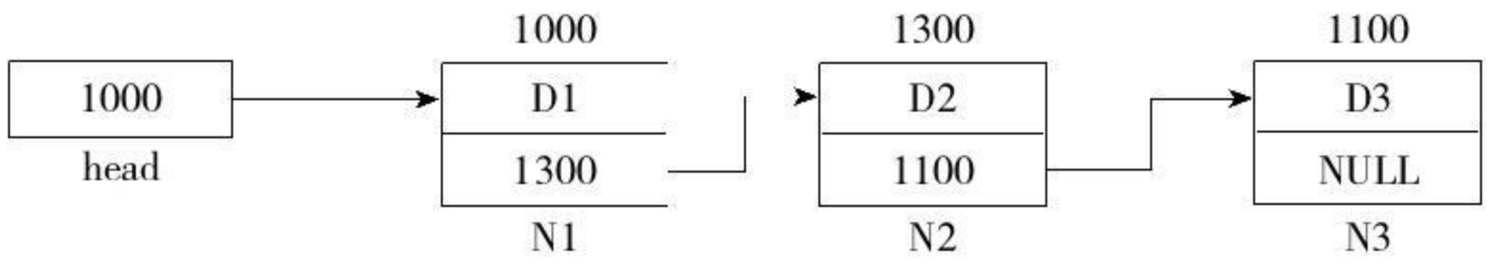


图 10-8　链表示意图

图 10-8 是链表的基本示意图，链表存在一个头指针 head，该结点没有数据域，只存放第一个含有数据结点的地址，在图中 head 存放了结点 N1 的地址 1000，显然，通过 head 可以访问 N1，N1 的数据域存放 D1，指针域存放下一结点 N2 的地址 1300，由此可以访问 N2…。最后一个结点的指针域存放 NULL 表示是链表的最后一项。

在链表结构中，结点可以动态地插入、删除、追加等。如要删除结点 N2，找到 N2 的前一个结点 N1，将 N2 的指针域的值 1100 赋给 N1 的指针域，N2 就被从链表中断开，而 N3

链接到 N1 之后，如图 10 - 9 所示。最后释放 N2 所占的内存，N2 就被删除了。

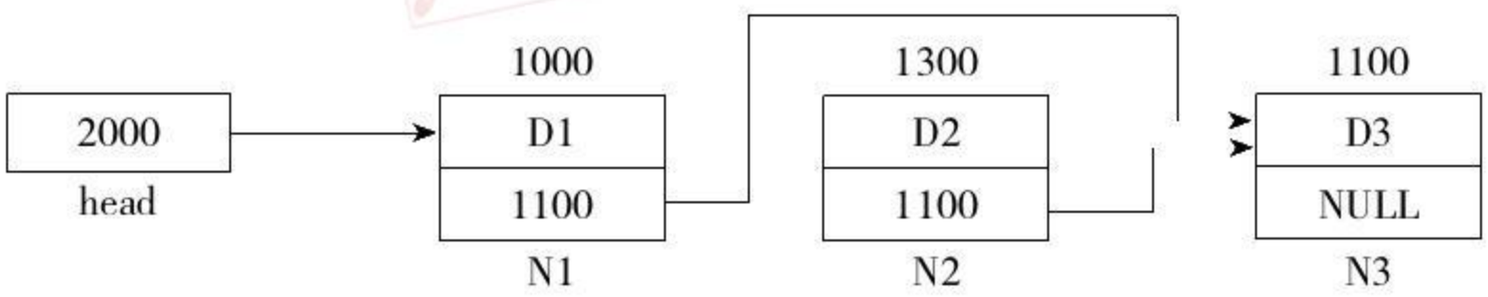


图 10 - 9　链表删除示意图

链表中每个结点的结构是相同的，包含了类型一致的数据域和指针域。指针域指向数据的类型与当前结点的类型一致。要实现如此的数据结构，必须定义一种结构体，结构体中必须有一个成员项，是指向同类型结构体的指针，称为引用自身的结构体。如：

```
struct  student{
            int  num;
            float  score;
            char  *name;                    /*三项数据域*/
            struct  student  *next;         /*指向同类型结构体的指针，指针域*/
           };
```

其中 next 是指向同类型结构体的指针，作为数据结点的指针域。

2. 内存分配函数

链表数据需要动态内存分配，因此不能像数组一样在使用之前说明。链表的相关操作必须允许程序在需要时灵活地分配内存，在不需要时也可以释放占用的内存。C 语言提供了一系列标准函数可以实现要求的功能。函数的原型定义在 stdlib. h 或 alloc. h 头文件中。下面介绍三个函数：

(1)malloc 函数

函数的原型声明为：

```
void *malloc(unsigned size);
```

函数的功能：

分配大小为 size(函数参数，最大 65535)字节的内存单元，并返回所分配内存单元的首地址，该地址是 void 类型，也就是说地址指向一个物理字节。分配内存可能成功，也可能失败。成功时，返回所分配内存单元的首地址；失败时，则返回 NULL(空)。

如：利用 malloc 函数分配一个整型单元，并赋值为 5。

```
int *p;
...
p=(int *)malloc(sizeof (int));
*p=5;
```

虽然分配了 int 类型所占字节数的内存单元，但返回的是 void 类型的地址，通过强制类型转换后变成了 int 类型的地址。

(2)calloc 函数

函数的原型声明为：

```
void *calloc(unsigned n , unsigned size);
```

函数的功能是分配 n 个 size 大小的连续内存单元。如果分配成功，返回所分配内存单元的首地址，类型为 void；如果分配失败，返回 NULL。

如：为有 10 个的浮点型数据分配内存：

```
float   *p;
...
p=(float *)calloc(10,sizeof (float));
```

(3)free 函数

函数的原型声明为：

```
void   free(void *ptr);
```

函数的功能：

释放 ptr 指向的由 calloc 或 malloc 分配的内存空间。函数无返回值。释放后的空间可以再次被使用。

3. 链表的基本操作

链表的操作主要包括如下几种：

① 建立并初始化链表；

② 遍历访问链表；

③ 删除结点。

下面将结合一个具体的实例，介绍链表的基本操作。主程序完成链表的建立与初始化，通过调用函数 append ( )可以在链表尾部追加结点；通过调用函数 showlist ( )，遍历显示链表的各结点的数据内容；通过调用函数 cdelete ( )，删除链表中的特定的结点。

(1)链表的建立

带有头指针链表的建立包括以下基本步骤：

① 定义引用自身的结构体；

② 说明指向头结点的指针 head 以及指向尾结点的指针 tail；

③ 初始化链表。初始化链表的主要目的是为链表建立头结点。基本步骤，先为头结点分配内存(数据域为空)，指针域指向 NULL(只有一个结点)。然后将 head、tail 指向头结点。

【例 10.8】　链表举例。

```
#include <stdio.h>
#include <stdlib.h>
void showlist (struct   slist *  );
void cdelete (struct   slist *  );          /*函数原型声明*/
struct   slist
         {
           int num;
           char name[10];
           struct slist *nextp;
         };                                 /*定义结构体类型,包含指向自身类型的结构体指针*/
struct   slist   *append ( struct slist *,int);
```

```
void main ( void)
{
   int iTmp;
   struct  slist *head, *tail, *find;/*说明头、尾指针以及删除结点时所用的指针 find*/
   head = tail = find=(struct slist*) malloc (sizeof ( struct  slist ));
   head->nextp = NULL;                    /*为建立头结点分配内存并初始化链表*/
   while(1)
   {
     printf ("INPUT:");
     scanf ("%d",&iTmp); /*输入结点的 num 成员项,存放在 iTmp 中,值 0 表示结束*/
     if (iTmp = = 0)
        break;
     else
        tail = append ( tail , iTmp);        /*调用函数在尾部插入结点*/
   }                                          /*循环追加结点*/
showlist (head);                              /*调用函数遍历显示链表的内容*/
scanf("%d",&iTmp);                            /*输入要删除结点的 num 成员项的内容*/
while (find->nextp! = NULL)
{                              /*循环比较 find 下一个结点的 num 成员项与输入是否相同*/
   if (find->nextp->num== iTmp)
   {
      cdelete (find);                         /*找到要删除的结点,调用函数删除结点*/
      break ;                                 /*终止循环*/
   }
   find= find->nextp ;          /*如果没有找到要删除的结点,指针指向下一个结点。*/
}
if (find->nextp == NULL)
   printf ("Not found\n" );
else
   showlist (head);                           /*显示删除的结果*/
}
```

(2)在链表尾部追加结点函数

在尾部追加结点函数的定义如下:

```
struct  slist  *append(struct  slist  *ptr,  int n)
{
   struct  slist  *p;
   p=(struct  slist  *)malloc (sizeof(struct  slist ));     /*为新结点分配内存*/
   p->num=n;
   gets(p->name);                             /*建立数据域*/
```

```
    p->nextp=NULL;
    ptr->nextp=p;
    return (p);
}
```

函数的第一个参数 ptr 是指向结构体 struct slist 的指针，传递的是结点的尾地址 tail；第二个参数传递的是主函数中输入 iTmp，对应的是结点的 num 成员项。尾部追加结点首先为结点分配内存，然后输入数据域中的 name 成员项，将指针域赋为 NULL，表示新建的结点是尾结点。最后将通过函数参数 ptr 指向的原尾结点的指针域指向新的结点。将新追加结点的地址返回赋给尾结点指针。

(3)显示链表的内容函数

```
void  showlist(struct  slist  *pshow)
{
  struct  slist  *p;
  p=pshow->nextp;                          /*p 指向头结点的下一个结点*/
  while (p! =NULL)                         /*没有到尾结点继续循环*/
  {
    printf("%9d,%s\n",p->num,p->name);     /*输出当前结点数据域的内容*/
    p=p->nextp;                            /指向下一个结点*/
  }
}
```

函数的参数是指向结构体的指针，传递的是头指针 head。函数中说明一个结构体指针，通过该指针不断访问各个结点，并显示数据域的内容。

(4)删除接点函数

```
  void  cdelete(struct  slist  *pcdel)
  {
    struct  slist  *p;
    p=pcdel->nextp;                          /*说明结构体指针并指向要删除的结点*/
    pcdel->nextp=pcdel->nextp->nextp;        /*断开链接*/
    free (p );                               /*释放删除结点所占的内存*/
  }
```

函数的参数是指向结构体的指针，传递要删除结点的前一个结点的地址。删除结点先将上一个结点的地址指向要删除结点的下一个结点，通过 pcdel->nextp=pcdel->nextp->nextp 完成。然后释放删除结点所占内存。

除此之外，还可以在链表的任意位置插入结点等，在此，就不再一一描述了。

10.2　共用体

与结构体类型类似，共用体也是一种由不同数据类型构造出的构造类型。但与结构体

不同的是,结构体的每个成员项都有独立的内存空间,而共用体类型的每个成员项使用公共的内存空间,只能对一个“活的(active)”成员项操作。而改变一个成员项的值,将修改其他成员项的内容。

10.2.1 共用体类型的定义

共用体类型定义的一般形式为:

```
union  共用体名{
        type  成员项 1;
        type  成员项 2;
              …
              …
        type  成员项 n;
        };
```

例如:

```
union  exam {
            int a ;
            float  b ;
            char  c ;
           };
```

共用体定义需要确定共用体的类型名,以及共用体类型中各个成员项名和成员项的类型。例中,定义了名为 exam 的共用体类型,共用体类型中包含三个成员项:整型的 a、浮点型 b、字符型 c。

10.2.2 共用体变量的说明

可以用共用体类型说明变量、数组、指针。说明的方式与结构体类似,可以采用三种方式:定义后说明、定义时说明、无名共用体定义时说明。例如:

```
union  exam  {
              int  a ;
              float  b ;
              char  c ;
             };          /*定义共用体类型*/
union exam x , *px , y[10];   /*说明共用体变量 x,指向共用体的指针 px,共用体数组 y*/
```

当说明共用体变量后,编译系统将按共用体数据类型的特点为共用体变量分配内存单元。由于共用体变量的所有成员项公用内存单元,因此,系统将以内存分配量最大的成员项为标准分配内存,其他的成员项则共用这段内存。如例中的 x 是一个共用体变量,成员项 b 是浮点型,所占内存单元最多。编译系统为共用体变量 x 分配四个字节的内存单元供成员项 b 使用,成员项 a 是整型,共用 b 的前两个字节,c 是字符型,共用 b 的头一个字节。如图 10-10 所示。

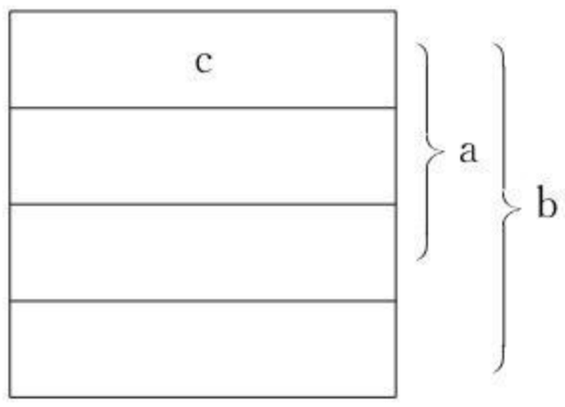


图 10-10　共用体变量内存的分配

10.2.3　共用体变量的引用

共用体变量的引用是指引用其成员项，引用的方式与结构体变量的引用方式相同，可以通过变量引用，也可以通过指针引用。引用的格式如下：

共用体变量名．成员项名

共用体指针名－＞成员项

例如：

```
x.a=5;                  /*共用体变量 x 的 a 成员项赋值为 5*/
px->c='a';              /*共用体指针 px 指向的共用体变量的成员项 c 赋值为'a'*/
```

共用体变量的成员项具有同类型简单变量的一切特征，因此，引用时可以当变量使用。由于共用体变量的各成员项共用同一内存区域，因此，当一个成员项改变时，其余的成员项也会改变。

【例 10.9】 共用体举例。

```
#include <stdio.h>
void  main (void)
{
  union  exam {
                int  i;
                char  str[2];
               } x, *px;
  px=&x;                  /*共用体指针 px 指向共同体变量 x*/
  x.i=25154;
  printf ("i=%xh\n", x.i);
  printf ("str[0]=%xh,str[1]=%xh\n", px->str[0], px->str[1] );
  printf ("str[0]=%c,str[1]= %c\n", x.str[0], x.str[1]);
}
```

程序运行结果，如图 10-11 所示。

图 10-11　例 10.9 程序运行结果

在程序当中共用体变量 x 占两个字节,其中 str[0]共用 i 的低字节,str[1]共用 i 的高字节,如图 10-12 所示。对 i 成员项赋值按照共用关系 str[0]、str[1]也获得了相应的值。程序最后以十六进制输出 x.i,str[0], str[1],在以字符格式输出 str[0], str[1]对应的正好是 B、b。

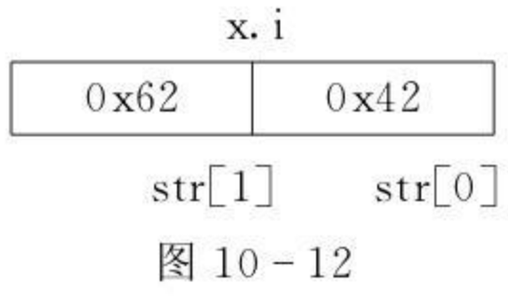


图 10-12

10.3 枚 举

在实际应用中,有些变量的取值被限定在一个有限的范围内。例如,一个星期内只有七天,一年只有十二个月等。这类数据的特征是有限状态的集合。每个状态,可以有一个名字,为此,C 语言提供了一种称为“枚举”的类型。在“枚举”类型的定义中列举出所有可能的取值以及每一个取值的名字,被说明为该“枚举”类型的变量取值不能超过定义的范围。

10.3.1 枚举的定义

枚举定义的一般形式为:

enum 枚举类型名{枚举元素表 };

如:enum day {SUN,MON,TUE,WED,THU,FRI,SAT} ; /* 定义该枚举名为 day,枚举值共有 7 个 */

枚举元素表反映了该枚举类型的变量所取值的集合。枚举元素如果不给值,自动取 0～n－1整数值(n 是枚举元素个数),如例中的 SUN 是 0,MON 是 1,…SAT 是 6;在定义枚举元素表时,可以对某个枚举元素赋值,其后元素的值将按顺序自动加一递增。

10.3.2 枚举变量的说明

与结构和共同体类似,枚举变量也可用不同的方式说明,即先定义后说明,同时定义说明或直接说明。

1. 先定义后说明

```
enum day{ SUN,MON,TUE,WED,THU,FRI,SAT };
enum day day1,day2,day3;
```

2. 同时定义说明

```
enum day{ SUN,MON,TUE,WED,THU,FRI,SAT } day1,day2,day3;
```

3. 直接说明

```
enum { SUN,MON,TUE,WED,THU,FRI,SAT } day1,day2,day3;
```

day1,day2,day3 是三个 enum day 类型的枚举变量,每个枚举变量只能取该类型中的一个元素的值。

10.3.3 枚举变量的引用

枚举变量不是简单的变量,其取值是枚举元素,在引用时应注意以下规则:

(1)枚举元素是常量,不是变量,不能在程序中用赋值语句再对它赋值。例如对枚举类

型 day 的元素再作以下赋值：MON＝2；是错误的。

(2)只能把枚举元素名赋给枚举变量，不能把元素对应的数值直接赋给枚举变量。如：day1＝MON；是正确的。而 day1＝1 是错误的。如果要赋枚举元素的对应的值可以通过强制类型转换，如：day1＝(enum day)1 赋值。

【例 10.10】 枚举举例。

```
#include<stdio.h>
void  main(void)
{
  enum day{ SUN,MON,TUE,WED,THU,FRI,SAT } x;
  scanf("%d",&x);/*以整型方式输入枚举变量 x*/
  switch(x)
  {
      case  MON:
      case  TUE:
      case  WED:
      case  THU:
      case  FRI: printf("工作日\n");break;
      case  SUN:
      case  SAT: printf("休息日\n");break;
      default:  printf("输入错误\n");
  }
}
```

程序运行结果，如图 10-13 所示。

图 10-13　例 10.10 程序运行结果

10.4　用户定义类型

C 语言不仅提供了丰富的数据类型，而且还允许由用户自己定义类型说明符，也就是说允许由用户为数据类型取“别名”。用户可以通过 typedef 给已经存在的系统类型或用户构造的类型重新命名。格式如下：

typedef　原类型名　用户定义类型名；

常用的用户定义类型主要有三种应用：用户定义基本类型，用户定义数组类型，用户定义的结构体或共用体类型。

10.4.1　基本类型定义

int 是整型变量的类型说明符。int 的完整写法为 integer，为了增加程序的可读性，可把整型说明符用 typedef 定义为：

```
typedef  int  INTEGER;
```

这以后就可用 INTEGER 来代替 int 作整型变量的类型说明了。例如：

```
INTEGER a,b;
```

它等效于：

```
int a,b;
```

10.4.2 数组类型定义

用 typedef 定义数组类型，可以使程序书写简单，而且使意义更为明确，因而增强了程序的可读性。

例如：

```
typedef  char  NAME[10];
```

表示 NAME 是字符数组类型，数组长度为 20。然后可用 NAME 说明变量，如：

```
NAME n1,n2,n3,n4;
```

完全等效于：

```
char n1[10], n2[10], n3[10], n4[10];
```

10.4.3 结构类型定义

结构体类型的一般定义形式为：

```
typedef  struct {
                 long  num ;
                 float  score ;
                } PERSONDOC ;
```

通过用户定义类型，将结构体的类型名定义成 PERSONDOC。可以用 PERSONDOC 说明结构体变量、数组、指针，如：

```
PERSONDOC  a , b , *p ;
```

在用户定义类型中用户定义的类型名一般用大写表示，以便于区别。

10.5 例题精解

【例 10.11】 有 5 个学生，每个学生的信息有学号、姓名和三门课的成绩，输出三门课的总平均分以及所有成绩中最高成绩所对应学生的全部信息。

```
#include <stdio.h>
void main(void)
{struct student
   { long num;
     char name[10];
     int score[3];
   }st[5]={{1001,"wang",67,75,88},{1002,"li",83,92,95},{1003,"zhao",56,82,79},
           {1004,"han",78,87,79},{1005,"qian",69,79,81}};
 int i,j,max,maxi;
 float aver[3]={0};
 for(j=0;j<3;j++)
```

```
    { for(i=0;i<5;i++)
          aver[j]+=st[i].score[j];
      aver[j]/=5;
    }
  max=st[0].score[0];
  for(i=0;i<5;i++)
    for(j=0;j<3;j++)
      if(st[i].score[j]>max){max=st[i].score[j];maxi=i;}
  printf("The averages of courses are:\n");
  for(i=0;i<3;i++) printf("%6.1f",aver[i]);
  printf("\n");
  printf("The informations of the student with maximal score:\n");
  printf("No.     Name    scor1   scor2   score3\n");
  printf("%ld%8s",st[maxi].num,st[maxi].name);
  for(j=0;j<3;j++)
    printf("%8d",st[maxi].score[j]);
  printf("\n");
}
```

程序运行结果,如图 10-14 所示。

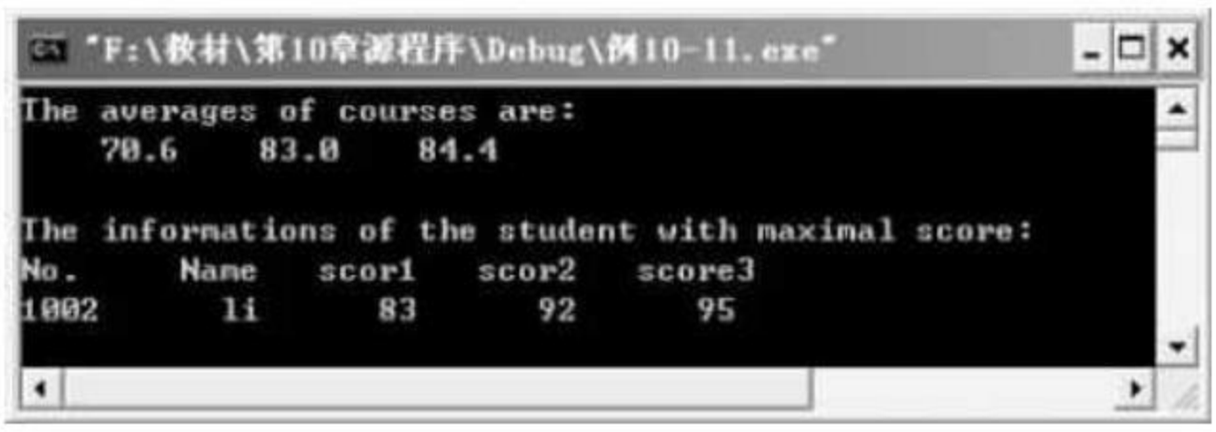


图 10-14　例 10-11 程序运行结果

【解析】 此程序中定义了一个结构数组,用来存放 5 个学生的相关信息。三门课的总平均分也定义了一个数组,找出最高成绩时应记录是哪个学生才能输出该学生的全部信息。

10.6　本章小结

本章主要介绍结构体、共用体和枚举类型和用户自定义类型。

结构体和共用体都是存放不同类型数据的集合。但结构体类型的定义只是给出了结构体的组织形式,并不占用存储空间,只有当定义了结构变量后,编译程序才会为结构变量分配相应的存储空间。结构变量占用的存储空间为各成员占用存储空间之和,而共用体、类型的变量所占空间是以最大长度成员项所占空间的大小为准,共用体变量在某一时刻只能存放其中一种成员。枚举类型可以将几个特定的数据组织在一起,形成枚举类型。

结构体、共用体都是由基本数据类型组合而成,组成这些数据类型的基本成份称为成员,对这些成员的操作需要使用成员运算符”.”或”->”。

习　　题

一、选择题

1. 已知:struct
{ int i;char c;float a;} ex;
则 sizeof(ex)的值是_____。
A) 4　　B) 5　　C) 6　　D) 7

2. 已知:
union{ int i; float a; char c ;} ex;
则 sizeof(ex)的值是_____。
A) 4　　B) 5　　C) 6　　D) 7

3. 设有以下说明语句
struct ex
{ int x ; float y; char z ;} example;
则下面的叙述中不正确的是_____。
A) struct 是结构体类型的关键字
B) example 是结构体类型名
C) x,y,z 都是结构体成员名
D) struct ex 是结构体类型

4. 有如下定义
struct person{char name[9]; int age;};
struct person class[10]={"Johu",17,
"Paul", 19,
"Mary",18,
"Adam", 16};
根据上述定义,能输出字母 M 的语句是
A) printf("%c\n",class[3]. mane);
B) printf("%c\n",class[3]. name[1]);
C) printf("%c\n",class[2]. name[1]);
D) printf("%c\n",class[2]. name[0]);

5. 以下结构体类型变量的定义中,不正确的是_____。
A) typedef struct aa
{ int n;
float m;
}AA;
AA td1;
B) #define AA struct aa
AA { int n;
float m;
}td1;
C) struct
{ int n;
D) struct
{ int n;

```
    float m;                    float m;
}aa;                        }td1;
struct aa td1;
```

6. 设有定义语句

```
enum team{ my,your=4,his,her= his+10};
```

则 printf(“%d,%d,%d,%d\n”,my,your,his,her);的输出是_____。

A) 0,1,2,3　　B) 0,4,0,10

C) 0,4,5,15　　D) 1,4,5,15

7. 若有如下定义,则 printf(“%d\n”,sizeof(them));的输出是_____。

```
typedef union{ long x[2];int y[4];char z[8];} MYTYPE;
MYTYPE them;
```

A) 32　　B) 16　　C) 8　　D) 24

8. 若有如下定义,则对 data 中的 a 成员的正确引用是_____。

```
struct sk{ int a;float b;} data, * p=&data;
```

A) (* p). data. a　　B) (* p). a

C) p－＞data. a　　D) p. data. a

9. C 语言共用体类型在任何给定时刻_____。

A) 所有成员一直驻留在结构中

B) 只能有一个成员驻留在结构中

C) 部分成员驻留在结构中

D) 没有成员驻留在结构中

二、填空题

1. “.”称为_____运算符,“－＞”称为_____运算符。

2. 若有如下定义语句,则变量 w 在内存中所占的字节数是_____。

```
union aa{ float x;float y;char c[6];};
struct st { union aa v;float w[5];double ave;} w;
```

3. 设有以下结构体类型说明和变量定义,则变量 a 在内存所占字节数是_____。

```
struct stud
{ char num[6];
  int s[4];
  double ave;} a, * p;
```

4. 以下程序用来输出结构体变量 ex 所占存储单元的字节数,请填空。

```
#include <stdio. h>
struct st
{ har name[20]; double score; };
void main(void)
{
   struct st ex;
   printf("ex size: %d\n",sizeof(_____));
```

```
}
```

三、应用题

1. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{ union { char c;char i[4];}z;
  z.i[0]=0x39;z.i[1]=0x36;
  printf("%c\n",z.c);
}
```

2. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
struct stru
{ int x;char ch;};
void main(void)
{ struct stru a={10,'x'}; func(a);
  printf("%d,%c\n",a.x,a.ch);
}
func( struct stru b)
{ b.x=100;b.ch='n';
}
```

3. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
union st
{ int i;char ch[2];}a;
void main(void)
{ a.ch[0]=13;a.ch[1]=0;
  printf("%d\n",a.i);
}
```

4. 阅读下列程序，写出运行结果。

```
#include <stdio.h>
struct stu
   { int x, * y;} * p;
   int a[]={15,20,25,30};
   struct stu aa[]={35,&a[0],40,&a[1],45,&a[2],50,&a[3]};
void main(void)
{
   p=aa;
   printf("%d"++p->x);
   printf("%d",(++p)->x);
```

```
    printf("%d\n",++(p->x);
}
```

四、编写程序题

1. 定义一结构体，成员项包括一个字符、一个整型。编程实现结构体变量成员项的输入，输出，并通过说明指针引用该变量。

2. 建立一结构体，其中包含学生的姓名、性别、年龄和一门课程的成绩。输出考分最高的同学的姓名、性别、年龄和课程的成绩。

3. 已知一个班有 20 个人，本学期有两门课程的成绩，求：

① 所有课程中的最高成绩，及对应的姓名、学号和课程编号。

② 课程 1，2 的平均成绩，并求出两门课程都低于平均成绩的学生姓名和学号。

③ 对编号 1 的课程从高到低排序(注意，其他成员项应保持对应关系)。

说明：要求定义结构体，第一成员项为学生姓名，第二成员项为学号，另外两个成员项为两门课成绩。①、②、③要求分别用函数完成。

4. 输入一字符串，用链表形式存储，每个结点的数据域存放一个字符。

5. 建立一个链表，每个结点包含学生的学号、姓名、性别、年龄和一门课程的成绩。要求输入一个学号，在链表中查找。显示结果后，如果该结点存在，删除该结点。

第11章 位 运 算

【教学提示】

C语言是中级语言,可以实现CPU的某些指令。位运算是CPU的一类运算,C语言通过位运算可以对二进制的位进行所需的运算,从而实现程序设计中的特殊功能。

【核心概念】

二进制的位 数据的表示方法 逻辑位运算 移位运算 位域

11.1 位运算的概念

前面所涉及的C语言的各种运算都是数据对象作为整体参加的运算。数据对象整体上表示一个完整的数据。而有时程序中要求对数据对象在二进制位(bit)一级进行运算,也就是对数据的某一位或某几位进行特殊的处理。C语言提供了完成这类按位运算的基本功能,称为位运算,位运算使得C语言也能像汇编语言一样对二进制位进行操作。

由于位运算的特殊性,位运算的运算对象只能是整型和字符型数据。

11.2 位运算

C语言提供了两类六种位运算。第一类是逻辑位运算,第二类是移位运算。逻辑位运算是本位相关的运算,运算结果只与对应的二进制位的值相关,而与高一位或低一位的二进制位的运算结果无关。

11.2.1 逻辑位运算

逻辑位运算共有四个运算符,运算符及其所表示的运算关系,见表11-1所列。

表11-1 逻辑位运算符

运算符	运算对象	运算关系
&	双目	逻辑位“与”运算
¦	双目	逻辑位“或”运算
^	双目	逻辑位“异或”运算
~	单目	逻辑位“非”运算

1. 逻辑位“与”运算

逻辑位“与”运算是运算对象的对应位按“与”的关系进行运算。逻辑“与”运算的运算规则见表11-2所列。显然只有对应位都是1时,运算结果是1,其余为0。

表 11-2　逻辑与运算的运算规则

a(bit)	b(bit)	a&b(bit)
0	0	0
0	1	0
1	0	0
1	1	1

从表中可以看到,0 与 x(bit),其结果为 0;1 与 x(bit),其值为 x。逻辑与运算可以使某些位清 0。其余的位维持原值。

【例 11.1】　取整型变量 a 的低字节,高字节清 0。

```
#include <stdio.h>
void main(void)
{
   int   a=3421;
   a= a & 0x00ff;
   printf("%d",a);
}
```

程序运行结果,如图 11-1 所示。

图 11-1　例 11.1 程序运行结果

从二进制的的角度看:

```
a=          0 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1     (3421 的补码表示)
        &   0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
-------------------------------------------------
            0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1     (93 的补码)
```

2. 逻辑位“或”运算

逻辑位“或”运算是运算对象的对应位按“或”的关系进行运算。逻辑位“或”运算的运算规则见表 11-3 所列,显然只要对应位中有一个 1,运算结果是 1,其余为 0。

表 11-3　逻辑位“或”运算的运算规则

a(bit)	b(bit)	a | b(bit)
0	0	0
0	1	1
1	0	1
1	1	1

逻辑位“或”运算中,显然 0 或 x(bit),结果是 x,1 或 x,结果是 1。因此,通过位“或”运算可以使某些位置 1。

【例 11.2】　将数值 3(字节型),转换为字符‘3’。

```
#include <stdio.h>
```

```
void main(void)
{
   int   a=3;
   char   ch;
   ch= a|0x30;
   putchar(ch);
}
```

图 11-2　例 11.2 程序运行结果

程序运行结果，如图 11-2 所示。

从二进制的角度，运算的过程如下：

```
a 的低字节        0 0 0 0 0 0 1 1
                  |
                  0 0 1 1 0 0 0 0
-------------------------------------
ch                0 0 1 1 0 0 1 1   (3 的 ASCII 码)
```


3. 逻辑位“异或”运算

逻辑位“异或”运算是运算对象的对应位按“异或”的关系进行运算。逻辑位“异或”运算的运算规则见表 11-4 所列，显然运算对象相异，结果是 1，运算对象相同，结果是 0。

表 11-4　逻辑异或运算的运算规则

a(bit)	b(bit)	a ^ b(bit)
0	0	0
0	1	1
1	0	1
1	1	0

逻辑位异或运算中，显然 0 异或 x(bit)，结果是 x，1 异或 x，结果 x 取反。因此，通过异或运算可以使某些位置取反，其余的位保持不变。

【例 11.3】　利用异或对数据加密。

```
#include <stdio.h>
void main(void)
{
   char   ch='a';
   ch=ch^5;
   putchar(ch);
   ch=ch^5;
   putchar(ch);
}
```

图 11-3　例 11.3 程序运行结果

程序运行结果，如图 11-3 所示。

从二进制的角度，运算的过程如下：

ch	1 1 0 0 0 0 1	‘a’的 ASCII
^	0 0 0 0 1 0 1	
ch	1 1 0 0 1 0 0	第一次异或后 ch 的值
^	0 0 0 0 1 0 1	
ch	1 1 0 0 0 0 1	第二次异或后 ch 的值，又恢复成’a’的 ASCII 码值。

利用异或的特点可以对数据进行简单的加密，可以先将数据异或一个值(密钥)加密，然后传输或存储数据，需要解密时在异或相同的值，数据即可恢复。

4. 逻辑按位取反运算

取反运算是单目运算。运算结果是运算对象按位取反(0 变成 1，1 变成 0)。如：

```
int a=123;
a=      0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1
~a      1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0
```

11.2.2　移位运算

移位是运算对象的二进制位整体向左或向右移动。左移运算符是<<，右移运算符是>>，移位运算是双目运算。

1. 左移运算

左移的功能把运算符<<左边运算对象的各二进位全部左移运算符右边的数所指定移动的位数，移位过程当中高位丢弃，低位补 0。例如：

```
int  b, a=123;
b=a<<3;
```

a 的各二进位向左移动 3 位。如二进制表示的 a 的值是 0000000001111001，左移 3 位后赋给变量 b，b 的值用二进制表示 0000001111011000(十进制值 984)。左移 1 位，相当于移位对象乘 2。如果移动的结果在表示范围内，移位实现的乘法要比算术实现的乘法的速度快。如，通过移位实现 a×5。

```
int  b,a=123;
b = a<<2;          /*b 的值是 a×4*/
b=b+a;             /* b 的值是 a×5*/
```

2. 右移运算

右移运算的功能是运算符 >>左边运算对象的各二进制位全部右移运算符右边的数所指定的移位数。例如：

```
int a=3456;
a>>5;                    /*表达式的结果是 a 的所有二进制向右移 5 位*/
```

移位过程中低位被舍弃。高位补入的处理方法根据移位对象的类型不同有一定的差别。对于整型有符号数，在计算机内部采用补码存储形式，补码的编码中，最高位代表符号，最高位是 0，表示正数，最高位是 1，表示负数。因此，多数 C 语言系统中有符号数在右移的过程中保持移位结果与移位前的符号是一致的，所以，右移时原来的最高位(符号位)自补移位后的最高位，这种移位方式称为“算术移位”。

```
int a=- 4;
```

a>>1;

a 的二进制补码表示为：

b15														b0
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

a 右移一位的结果：

b15														b0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

移位后的最高位，用原最高位(符号位)填补。

多数 C 语言系统，对于无符号数，由于二进制的每一位表示的都是数值，如果移位的对象是无符号数，右移最高位始终补 0，这种移位称为“逻辑移位”。例如：

unsigned a=0xfffc;

a>>1;

a 的二进制表示为：

b15														b0
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

a 右移一位的结果：

b15														b0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0

移位后的最高位，用 0 填补。

右移一位，相当于有符号数或无符号数除 2，如果移动的结果在表示范围内，移位实现除法要比算术实现除法的速度快。

逻辑位运算中的双目运算符 &、|、^，以及移位运算都可以与赋值运算符结合成运算赋值 op=。如：

a^=5; /*表示的运算关系 a=a^5*/

11.3 位 域(位段)

C 语言的抽象数据类型都是以一个或几个字节为基本的存储单位。但在某些场合，所表示的信息并不需要占用一个完整的字节或几个字节，而只需用一位或几位二进制。例如灯的开关状态，只要一位二进制 0 和 1 两种状态就可以表示了。为了节省存储空间，并使表示和处理简单，C 语言提供了一种特殊的数据结构，称为“位域”(位段)。

所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域所占的二进制位数。每个域定义一个域名，在程序中可以对域名进行操作，这样就可以把几个不同的对象用一个字节中划分出来的几个二进制位域来表示。

11.3.1　位域的定义及位域变量的说明

位域的定义采用结构的定义方式，在位域的定义中需要说明位域类型的名称，位域中每个域的名称以及所占的二进制位数。

位域定义与结构定义类似，其一般形式为：

```
struct  位域类型名
  {
    type  位域名 1:长度 1;
    …
    type  位域名 n:长度 n;
  };
```

在位域的类型定义时，type 可以是 int、unsigned、signed 中的一种。

例如：

```
struct bitf
{
  int a:8;
  int b:2;
  int c:6;
};
```

位域变量的说明与结构变量说明的方式相同。可采用先定义类型后说明，定义的同时说明或者直接说明三种方式。例如：

```
struct bitf
{
  int a:8;
  int b:2;
  int c:6;
}x1;
```

x1 为 bitf 类型变量，一共占两个字节。其中位域 a 占 8 位，位域 b 占 2 位，位域 c 占 6 位。

关于位域的几点说明：

(1)一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：

```
struct bs
{
  unsigned a:4
  unsigned :0              /* 空域 */
  unsigned b:4             /* 从下一单元开始存放 */
  unsigned c:4
}
```

在这个位域定义中，a 占第一字节的 4 位，后 4 位填 0 表示不使用，b 从第二字节开始，

占用 4 位,c 占用 4 位。

(2)由于位域不允许跨两个字节,因此位域的长度不能大于一个字节的长度,也就是说不能超过 8 位。

(3)位域可以没有位域名,无名的位域只是用来填充或调整位域的位置。无名的位域是不能使用的。例如:

```
struct k
{
  int a:1
  int   :2                    /*无名位域,该2位不能使用*/
  int b:3
  int c:2
};
```

从以上分析可以看出,位域在本质上就是一种结构类型,不过其成员是按二进位分配的。

11.3.2　位域变量的使用

位域的使用和结构成员的使用相同,其一般形式为:

位域变量名.引用的位域名

【例 11.4】 位域的应用举例。

```
#include <stdio.h>
void main(void)
{
  struct bitf
  {
    unsigned a:1;
    unsigned b:3;
    unsigned c:4;
  }x,*px;                     /*定义位域结构,并说明位域变量bit以及指向位域的指针。*/
   x.a=1;
   x.b=7;
   x.c=15;
   printf("%d,%d,%d\n",x.a,x.b,x.c);
   px=&x;                     /*让px指向变量x*/
   px->a=0;                   /*通过指针,将px指向变量x的位域a赋值为0*/
   px->b&=3;                  /*通过指针,将px指向变量x的位域b位与3再赋值给位域b*/
   px->c|=1;                  /*通过指针,将px指向变量x的位域c位或1再赋值给位域c*/
   printf("%d,%d,%d\n",px->a,px->b,px->c);
}
```

程序运行结果,如图 11-4 所示。

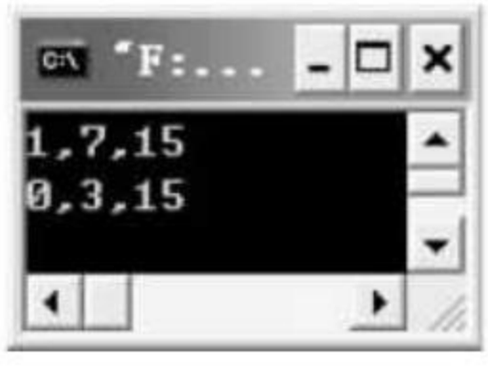


图 11-4　例 11.4 程序运行结果

【解析】 程序中定义了位域结构 bitf,在结构中有三个位域为 a,b,c。同时说明了位域变量 x 和指向位域类型的指针 px。接下来,分别给三个位域赋值(应注意赋值不能超过该位域的允许范围),并以整型量格式输出三个位域的内容。

11.4　本章小结

本章主要介绍了有关位运算的运算符、位域的定义及使用。位运算符可以与赋值符在一起组成复合赋值符,如|=,&=,^=,~=,<<=,>>=等。

利用位运算可以完成汇编语言的某些功能,如移位、清零、置 1 等。位域在本质上也是结构类型,只不过其成员按二进制位分配内存,它的定义、说明、使用都与结构类型相同。位域的使用,实现了数据的压缩,节省了空间,提高了程序的效率。

习　　题

一、选择题

1. 下列表达式的值(设 int a=2),等于 1 的是_____。

A)~a&a　　B)~a||a　　C)a>a　　D)a!=a

2. 表达式~a&b||c<d 的运算顺序是_____。

A)~,<,&,||　　B)~,&,||,<　　C)~,&,<,||　　D)~,||,&,<

3. 表达式 0x13&0x17 的值是_____。

A)0x17　　B)0x13　　C)0xf8　　D)0xec

4. 设 a=3,b=2,则表达式 a^b>>2 的值是_____。

A)00000011　　B)00000110　　C)00000100　　D)00000010

5. 下列程序段运行后变量 z 的二进制值是_____。

```
char x=3,y=6,z;
z=x^y<<2;
```

A)00010100　　B)00011011　　C)00011100　　D)00011000

二、填空题

1. 已知:a=15,b=20;则表达式(a&b)&b||b 的结果为_____。

2. 以 16 进制的形式写出 0xa1b2&0x1a2b 的运算结果_____。

3. 以 16 进制的形式写出!0xa1b2 的运算结果_____。

4. 设二进制数 A 为 00101101,若想通过异或运算 A^B 使 A 的高 4 位取反,低 4 位不变,则二进制 B 应是_____。

三、应用题

1. 有下面程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{
   int a=123,b=55;
   printf("a=%d\n",a);
   a=a^b;
   printf("a=%d\n",a);
   a=a^b;
   printf("a=%d\n",a);
}
```

2. 有下面程序，写出运行结果。

```
#include <stdio.h>
void main(void)
{
   unsigned a,b,c;
   int n;
   scanf("a=%o,n=%d",&a,&n);
   b=a<<(16-n);
   c=a>>n;
   c=c|b;
   printf("%o\n%o",a,c);
}
```

四、编写程序题

1. 编写程序实现输出整型变量 a 的高字节的值与低字节的值。

2. 输入一个有符号整型存放的变量 a 中，求 a 的补码，存放到无符号整型变量 b 中，用十六进制格式输出 b。

3. 编写程序通过异或运算对输入的字符串加密，密钥通过键盘输入，并验证结果。

第 12 章　文　　件

【教学提示】

输入/输出是程序和外设交换数据的重要手段。本章介绍存储在外设上的数据文件与程序的数据交换。包括文件的读、写、定位等基本操作的实现；文件的打开、关闭、读、写、定位等函数的调用形式以及文件操作在程序设计中的应用方法。

【核心概念】

文件的概念与分类　文件的读写方式　文件的打开与关闭　文件的读写与定位

12.1　文件概念

前面章节介绍的程序输入输出的外部设备都是控制台，即输入时的键盘，输出时的显示器。控制台在输入输出是实时的，不便于数据的修改、维护、保存和交换。而事实上计算机的输入输出设备是多种多样的，除了控制台外，主要的输入输出设备是磁盘等。

文件是一组存储在外部介质上的相关数据的集合，可以是源文件、目标程序文件、可执行程序，也可以是待输入的原始数据，或是一组输出的结果。文件用文件名标识。文件名是引用文件的唯一标志。

C 语言可以将输入输出设备映射成文件。文件可以分为磁盘文件、设备文件。磁盘文件是存储在磁盘上的数据集合，设备文件是指各种外部设备，如显示器、打印机、键盘等，设备文件有一个系统提供的专用名称标识，如打印机是 stdprn。以下“文件”不加说明特指磁盘文件。

12.1.1　文件的分类

按文件的存储形式、读写方式以及处理的方法的不同。文件可以分成如下几类：

1. 存储形式

从文件存储的形式来看，文件可分为文本文件和二进制文件。文本文件在磁盘中存放数据字符所对应的 ASCII 码。

例如，整型数 234 的存储形式为：

ASCII 码　　00110010 00110011 00110100

符　号　　2　　3　　4

由于文本文件的内容是字符，可以显示和编辑，便于维护和查看。

二进制文件是按二进制的编码方式来存放数据。例如，同样整型数 234 的二进制存储形式为：

00000000 11101010(两个字节表示的 234 的二进制补码)

2. 处理方法

根据文件读写时的处理方法，可以将文件分为：缓冲文件和非缓冲文件。缓冲文件是在磁盘和程序之间插入的一段称为缓冲区(Buffer)的内存区域，如图 12－1 所示。程序与磁

盘文件之间不是直接读写,而是通过缓冲区进行读写的。缓冲区的主要作用是匹配主机与外设的速度,提高数据的传送效率并实现数制的转换。现在的文件系统主要采用缓冲文件,非缓冲文件在文件与程序之间没有缓冲区,效率较低。

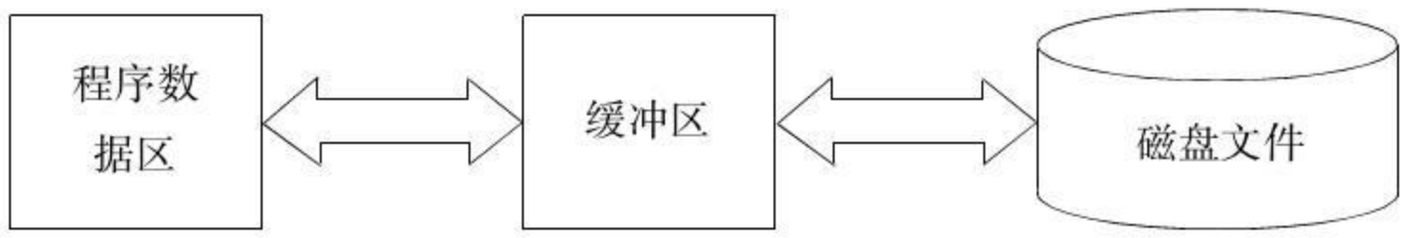


图 12-1 缓冲文件

3. 读写方式

按文件的读写方式可以将文件分为顺序读写文件、随机读写文件。顺序读写文件的读写按照数据在文件中的存储顺序进行,读写完当前数据后,下次读写自然是下一组数据,因此,无需在文件读写时定位。随机文件可以读写文件任意位置的数据,但必须在读写前定位到数据在文件的位置处。

12.1.2 文件的操作过程

在C语言中,文件独立于程序而存在,一个C程序对文件的操作所采用的步骤是基本一致的。所有文件的操作都要经过四个步骤:

1. 定义标准的文件指针

该指针所指向的结构体变量在文件的使用过程中始终保存文件的基本信息。

2. 打开文件

打开文件的目的是建立一个以文件名标识的磁盘文件与文件指针的联系,建立相应的缓冲区以及文件基本信息结构变量。

3. 读写文件

读写文件是文件操作的目的。读是指从文件输入数据到程序的数据区,写是指将程序数据区中的数据输出到文件。

4. 关闭文件

文件使用后要关闭文件,关闭文件的目的是释放文件打开时所占用的资源。

对于文件的打开、读写、关闭,C语言定义一簇标准的函数,函数的原型声明在头文件stdio.h当中。对于文件的应用,主要是掌握文件操作的各个函数的基本使用方法。

重点:文件是保存在磁盘上的信息组织形式。在C程序中使用的保存数据文件按存储格式分两种类型:文本文件和二进制文件。

12.2 文件指针

要使用一个缓冲文件,必须知道文件的必要信息。如文件的缓冲区地址、缓冲区中的剩余的字节数、文件的读写模式等。在缓冲文件系统中。每个打开的文件都在内存中开辟一段区域,这个区域是一个系统定义的结构体变量用于存放文件的有关信息。该结构体类型

由系统定义,采用用户定义类型方式,即:

```
typedef   struct { …/ * 内容 * / }FILE ;
```

显然,该结构体的用户定义类型名为:FILE。当打开一个缓冲文件的时候,系统为文件建立一块内存区域,用于保存反映文件信息的结构体变量,并返回该结构体变量的地址。用户要为打开的每个文件说明一个指向文件信息结构体变量的指针,打开文件时将结构体变量的地址赋给指针。文件指针在文件的读写过程中逻辑上代表了该文件。

文件指针的说明方式:

```
FILE   * fp1, * fp2,…, * fpn; / * fp1,fp2…fpn 是说明的文件指针 * /
```

12.3　文件的打开与关闭

在说明文件指针后,可以通过 C 语言定义的标准函数打开文件,然后对文件进行读写,最后再关闭文件。下面介绍文件的打开与关闭函数的使用方法。

12.3.1　文件的打开

fopen 函数的原型定义在头文件 stdio.h 中,用该函数可以打开文件。一般形式为:

```
FILE   * fp ;
fp = fopen ( filename , mode );
```

fopen 函数有两个参数,第一个参数 filename 是一个指向字符的指针,可以用字符数组名或字符串常量表示打开文件的文件名(包含盘符及路径);第二个参数 mode 也是一个指向字符的指针,是系统规定的字符串,用来表示文件的操作属性。

该函数返回包含文件信息的结构体变量的地址,将此地址赋给一个文件指针,文件指针在文件操作的整个过程中就表示打开文件的文件名所对应的文件。例如:

```
FILE   * fp ;
fp = fopen ("c:\\dos\\help.doc" ,"r" );
```

表示用“只读”方式打开 c:\dos 路径下的名为 help.doc 的文件

参数 mode 表示文件打开后读写文件的基本操作方式,mode 参数是系统定义好的字符串,每个不同的字符串表示一种特定操作模式,表 12-1 给出了字符串即对应的操作模式。

在打开文件的模式中,三个字母表示基本的模式,“r”(read)模式总是打开一个已经存在的文件,如果文件不存在打开时将出错。“w”(write)建立一个新文件,如果文件已经存在,那么先删除存在的文件,然后建立新文件。“a”(append)打开一个存在的文件,在文件的尾部追加数据。b 表示二进制文件,该项缺省表示文本文件。+表示将读写模式扩展为可读、可写。

fopen 打开文件有可能失败,比如用“r”模式打开一个不存在的文件。打开文件失败,fopen 函数将返回一个指向空指针 NULL。由于文件打开错误时,程序无法正确输入输出数据,也就是说程序无法继续执行,必须在打开文件时,判断是否出错,如果出错,输出错误信息,并终止程序运行。因此,一般用如下方式打开文件:

```
FILE   * fp ;
if (( fp=fopen( filename , mode ))==NULL )
```

```
{
    printf ("打开文件错误！\n" );
    exit (1);              /* 由 exit 函数终止程序运行,退回操作系统 */
}
```

打开文件后,如果判断文件指针的内容是 NULL,则打开文件失败,输出提示信息并终止程序运行。

表 12-1 mode 字符串对应的操作模式

字符串	文件类型	读写模式
“r”	文本文件	打开一个已存在的文件用于输入。
“w”		建立一个新的文件用于写,如文件已经存在,则先删除。
“a”		打开一个已存在的文件,用于在尾部追加数据。
“rb”	二进制文件	打开一个已存在的文件用于输入。
“wb”		建立一个新的文件用于写,如文件已经存在,则先删除。
“ab”		打开一个已存在的文件,用于在尾部追加数据。
“r+”	文本文件	打开一个已存在的文件,用于读/写。
“w+”		建立一个新文件,用于读/写。
“a+”		打开一个已存在的文件,用于在文件尾部读/写。
“rb+”	二进制文件	打开一个已存在的文件,用于读/写。
“wb+”		建立一个新文件,用于读/写。
“ab+”		打开一个已存在的文件,用于在文件尾部读/写。

12.3.2 文件的关闭

文件打开的目的是为了读写,当文件使用完毕后,应关闭文件。关闭文件主要有三个目的:第一,保证文件的数据不丢失,将缓冲区中的数据回写文件;第二,释放缓冲区;第三,断开文件指针与文件的联系,使关闭后的文件指针可以被用于打开其他文件。

C 语言定义了关闭文件的标准函数 fclose。函数的原型定义在头文件 stdio.h 中。

一般形式为:

```
fclose(fp);              /* fp 是文件指针 */
```

fclose 函数在关闭文件正确时返回整型值 0;关闭失败返回非 0。

12.4 文件的读写

文件打开后,可以通过系统定义的一系列标准函数实现对文件的读写,这些函数的原型都声明在头文件 stdio.h 中。分类介绍如下:

12.4.1　字符输入/输出函数

1. 字符输出函数 fputc

一般形式为：

fputc (ch , fp);　　/ * fp 是要写入数据的文件指针 * /

函数的功能是，写一个字符(或一个字节的数据)到 fp 对应文件的当前位置上。如果调用函数成功，则返回 ch 的值；如果失败，则返回值 EOF(系统定义的宏，值为－1)。

2. 字符输入函数 fgetc

一般形式为：

ch = fgetc (fp);　/ * fp 是可读文件的文件指针 * /

函数的功能是从 fp 对应文件的当前位置读一个字符，该文件必须是以读写方式或读方式打开的。如果调用成功返回读到的字符(赋值给 ch)；如果读到文件结束，返回 EOF (－1)。

3. feof 函数

一般形式为：

feof (fp);　　/ * fp 是文件指针 * /

函数的功能是判断文件是否结束。当返回值是 1 时，表示文件结束；当返回值是 0 表示文件未结束。

【例 12. 1】 将一个磁盘文件中的数据复制到另一个文件中去。

```
#include <stdio. h>
#include <stdlib. h>
void   main (void)
{
  FILE   * in , * out;                                  / * 定义文件指针 * /
  char   ch
  if ( ( in=fopen ("c:\\f1. txt","r"))==NULL)    / * 以读方式打开源文件 * /
  {
    printf ("Cannot   open   the   infile \n");
    exit (1);
  }
if (( out=fopen ("c:\\f2. txt","w" ))== NULL )/ * 以写方式打开目标文件 * /
{
  printf ("Cannot   open   the   outfile \n");
  exit(1);
}
while ((ch=fgetc(in))!=EOF)
  fputc (ch, out );
/ * 当源文件没有结束时，循环从源文件(in)读入字符并写入目标文件(out) * /
fclose (in);
```

```
    fclose (out);                                    /*关闭文件*/
}
```

程序运行结果,如图 12-1 所示。

(a) 源文件 f1.txt　　(b)目标文件 f2.txt

图 12-1　例 12.1 程序运行结果

12.4.2　文件的字符串输入/输出函数

这一对函数的主要作用是完成对文件的字符串输入/输出。

1. 字符串输入函数 fgets

一般形式为:

fgets(str, n,fp);

函数共使用三个参数,str 是字符指针或数组名,指明输入的字符串存放在内存中的首地址;n 是整型量,说明输入字符串的最大长度;fp 是文件指针,对应输入的文件。

函数的功能是从 fp 对应文件的当前位置,最多输入 n−1 个字符,在最后加'\0'后存放在 str 为首地址的内存中。由于输入的是字符串,在文件输入的过程中,如果遇到换行符或 EOF,输入即结束。函数正常调用,返回 str 的首地址,当出错或遇到文件结束标志时,返回 NULL。

2. 字符串输出函数 fputs

一般形式为:

fputs (str , fp);

函数参数 str 是字符指针或数组名,fp 是对应输出文件的文件指针。函数的功能是将首地址是 str 的字符串,输出到 fp 对应文件的当前位置,自动丢弃 str 后的'\0'。函数调用成功,返回值是 0,函数调用失败返回值是 EOF。

【例 12.2】 在打印机上打印磁盘文件的内容。

```
#include <stdio.h>
#include<stdlib.h>
#define   SIZE   256
void main ( int   argc, char  *argv[] )
{
   char buff[SIZE] ;
   FILE   *fpr, *fpw;        /*说明两个文件指针*/
   fpw=stdprn;               /*将标准设备文件打印机的文件指针 stdprn 赋给 fpw*/
   if (argc! =2)
```

```
{
   puts ("Input   filename:" );
   exit (1);
}
/* 磁盘文件的文件名是命令行参数,如果没有输入文件名,提示并终止程序运行 */
if (( fpr= fopen (argv[1],"r"))== NULL )
{
   printf("%s file cannot opened \n", argv[1]);
   exit (1);
}
/* 打开 argv[1]对应的文件,如果失败,提示错误,终止程序运行 */
while (fgets (buff , SIZE, fpr )! = NULL )
   fputs (buff , fpw );
/* 当文件没有结束时,循环输入字符串到 buff,并输出到 fpw 指向的标准打印机上 */
fclose (fpr) ; /* 关闭文件 */
}
```

在文件使用中,可以采用标准的设备文件,系统为每一个设备指定了一个标准的文件指针名称。在程序当中 fpw 指向了标准的打印设备 stdprn,stdprn 是打印机的标准文件指针名称。常见的标准设备名,见表 12-2 所列。

表 12-2　常见的标准设备名

名称	所指设备
stdin	标准输入设备,键盘
stdout	标准输出设备,显示器
stdprn	标准打印机

当需要通过文件的读写函数使用标准设备时,可以在文件指针的位置,用标准设备的指针名代替。例如:

```
fputs("Hello!",stdout);/* 在屏幕上显示“Hello!” */
```

12.4.3　文件的格式化输入/输出函数

1. 格式化输入函数 fscanf

一般形式为:

fscanf (fp , format , &arg1 , &arg2 ,…, &arg n);

函数由三个参数:

(1)fp 是输入文件的文件指针;

(2)format 是格式说明字符串,与 scanf 函数的使用方法相同;

(3)&arg1……&arg n 为输入变量的地址列表。

从 fp 指向的文件的当前位置,顺序读取 ASCII 码值,按照 format 规定的格式,转化成

各个变量对应的值,送入指定变量。

【例 12.3】 从名为 text.txt 的文件中读一字符串和一个十进制数,输出到显示器。

```
#include <stdio.h>
#include <stdlib.h>
void main (void)
{
   char  s[80];
   int   a;
   FILE   *fp;
   if (( fp=fopen ("c:\\text.txt" ,"r" ))== NULL)
   {
      printf ("Cannot   open   file");
      exit (1);
   }/*打开 text.txt 文件,如果出错,打印错误提示,终止程序运行*/
   fscanf ( fp ,"%s%d" , s ,&a );
   /*从文件中,以"%s %d"格式输入一个字符串和一个整型值给 s 和 a*/
   printf ("%s,%d\n", s, a );
}
```

程序运行结果,如图 12-2 所示。

(a)源文件 text.txt (b)输出结果

图 12-2 例 12.3 程序运行结果

在文件中的内容必须与格式说明串中规定的格式内容一致。比如例题中的输入文件当前位置的格式应是:…Hefei 123…。

2. 格式化输出函数

一般形式为:

fprintf (fp, format, arg1,…,arg n);

参数说明:

(1)fp 是输出文件的文件指针;

(2)format 是格式说明字符串,与 printf 函数的使用方法相同;

(3)arg1,arg2,…,arg n,输出参数表。

函数的功能是按指定的格式(format)将输出列表 arg1,arg2,…,arg n 的值转换成对应的 ASCII 码表示形式,写入 fp 文件的当前位置。例如:

```
fprint(fp,"%d,%x,%u",123,145,12);
```

12.4.4　文件的数据块输入/输出函数

1. fread 从文件中读入数据到内存缓冲区

一般形式为：

fread (buf, size, count,fp);

参数说明：

(1)buf,类型为 void 指针,内存中存放数据的首地址；

(2)size,类型为无符号整型,一次读取的字节数；

(3)count,类型为无符号整型,读取的由 size 表示大小的数据块的次数；

(4)fp 文件指针。

函数的功能是从 fp 指向文件的当前位置,读取 size 个字节,共 count 次,总字节数为 size×count,存放到首地址为 buf 的内存中。函数调用正确返回读取项数的值。

2. fwrite 从内存输出数据块到文件

一般形式为：

fwrite (buf, size, count, fp);

参数说明：

(1)buf,类型为 void 指针,内存中存放数据的数据区首地址；

(2)size,类型为无符号整型,写文件块的字节数；

(3)count,类型为无符号整型,写大小为 size 块的次数；

(4)fp,文件指针。

函数功能是从 buf 开始,分 count 次,每次 size 个字节,向 fp 指向的文件的当前位置写数据,共 count×size 字节。正确调用时返回 count。

显然,fread 函数和 fwrite 函数读写的最小单位是字节,而 fscanf 函数和 fprintf 函数的读写数据基本单位是以类型为单位的数据对象。因此,fread 函数和 fwrite 函数更适合处理二进制文件,而 fscanf 函数和 fprintf 处理的都是文本文件。

【例 12.4】 用一个结构体类型存放一个学生的基本情况,将一个班同学的基本情况数据定义成一个结构体数组,从键盘输入结构数组,将其写入文件 docu.dat 中。然后,从文件中读入数据,求出成绩的最大值。

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM 20
void main(void)
{
   struct stu{
          long iNum;
          char chpName[10];
          float fScore;
        } clas[NUM],max;
```

```
    FILE *stream;
    int i;
    float fMax;
    char str[20];
    stream = fopen("docu.dat","w");                    /*打开文件*/
    for(i=0;i<NUM;i++)
    {
      printf("\n输入第%d个人的姓名",i);
      gets(clas[i].chpName);
      printf("\n输入第%d个人的学号",i);
      gets(str);                                       /*以字符串方式输入学号*/
      clas[i].iNum=atol(str);      /*将字符串方式的学号转换成长整型*/
      printf("\n输入第%d个人的成绩",i);
      gets(str);
      clas[i].fScore=atof(str);
    }          /*循环输入45个人的学号、姓名、分数*/
    fwrite(clas, sizeof (struct stu), NUM, stream);
    /*求出结构体所占的字节数,将clas为首地址的NUM个结构数组元素写入文件*/
    fclose(stream); /*关闭文件*/
    stream = fopen("docu.dat", "r");
    /*重新打开文件,从头读文件*/
    fread(&max, sizeof (struct stu), 1, stream);
    /*读一个结构所占的字节,存放到结构体变量max*/
    fMax=max.fScore;
    for(i=1;i<NUM;i++)
    {
      fread(&max, sizeof (struct stu), 1, stream);
      if(max.fScore>fMax)
        fMax=max.fScore;
    }
      printf("%f\n", fMax);
      fclose(stream);
    }
```

12.4.5 整数输入/输出函数

1. 整数输入函数getw

一般形式为:

```
int  a;
a=getw ( fp );
```

函数的功能是从 fp 指向的文件中读一个整数(2 字节),整数由函数返回。该函数只适用于二进制文件。

2. 整数输出函数 putw

一般形式为:

putw(i,fp);

函数的功能是将整数 i 输出到文件 fp 之中。该函数适用于二进制文件。

【例 12.5】 将一个整数通过 putw 函数写入文件,关闭文件后再次打开,通过 getw 函数读一个整数在屏幕上显示。

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
   FILE *fp;
   int word;
   fp=fopen("text.bin", "wb");
   if (fp== NULL)
   {
      printf("Error opening file\n");
      exit(1);
   }                          /* 建立一个名为 text.bin 的二进制新文件,用于写 */
word = 94;
putw(word,fp);
fclose(fp);                   /* 将整型变量 word 写入文件后,关闭文件 */
fp = fopen("text.bin", "rb");
if (fp == NULL)
{
   printf("Error opening file");
   exit(1);
}                             /* 打开 text.bin,用于读 */
word = getw(fp);
   printf("%d",word);         /* 从文件中读一个整数,在屏幕上显示 */
}
```

12.5　文件的定位操作

以上介绍的文件读写函数都是按顺序读写,也就是说当完成一次读或写操作后,文件的当前位置会自动移动到下一位置,因而称为顺序读写文件。C 语言也允许在文件的任意位置读写数据,这种读写方式称为随机读写方式。要完成文件的随机读写,必须确定及修改文件读写的当前位置,C 语言定义了一系列函数,可以实现文件读写的定位以及修改。

1. 取文件当前位置 ftell 函数

一般形式为：

```
long   n;
n = ftell(fp);
```

fp 是文件指针。函数的功能是取文件的当前的读写位置，所谓读写位置指的是从文件开始处到当前读写位置的字节数(用长整型量表示)。函数调用正确返回文件当前读写位置，调用错误，函数返回－1L。

2. 改变文件指针的当前位置 fseek 函数

一般形式为：

```
fseek( fp, offset, from);
```

参数说明：

(1)fp 为文件指针；

(2)offset，类型是长整型，表示以 from 为起点移动的量相对值(字节数)；

(3)from 为移动的起始位置。

from 的取值是系统规定的，表 12－3 给出了 from 的值以及系统定义的宏名以及表示的在文件中的位置。

表 12－3　from 的值及其表示的位置

from 的值	from 的宏名	表示文件的位置
0	SEEK_SET	文件头
1	SEEK_CUR	读写的当前位置
2	SEEK_END	文件尾

函数的功能是将文件的读写位置以 from 为参照点，移动 offset 个字节。例如：

```
fseek( fp , 50L, SEEK_SET);
```

其含义是将读写指针移动到距文件开始后的 50 个字节处。

3. 置文件读写位置于开头 rewind( )函数

一般形式写：

```
rewind( fp ) ;
```

fp 是文件指针。函数的功能是将文件的当前位置移动到文件的开始处。

【例 12.6】 已知 30 个学生的一门分数，显示学号是单号的同学的成绩。

```
#include<stdio. h>
#include<stdlib. h>
#define   N   30
struct   Student   {
                    char   name[10];
                    int   no;
                    int   score ;
```

```
    } stud [N];
void main (void)
{
   int   i;
   FILE   * fp;
   if((fp=fopen("std. lst" ,"rb"))== NULL)
   {
      printf("Cannot open the file \n");
      exit(1);
   }
   for(i=0; i<n; i++)
   {
      fseek ( fp, i * sizeof( struct   student ), 0 );
      fread (&stud[i], sizeof (struct   student ), 1, fp );
      if (stud [i]. no%2)
         printf ( "name:%c no:%d score:%d\n",stud[i]. name , stud[i]. no ,
stud[i]. score );
   }
   fclose(fp);
}
```

12.6　文件的错误检测

C 语言提供了错误检测函数,可以检查文件读写时出现的错误。

1. 文件读写错误检测函数 ferror(fp)

一般形式为:

int errorcode;

errorcode = ferror(fp);

fp 文件指针。文件的每次读写调用,都产生一个是否错误的值,调用函数 ferror 的目的就是获得最近一次读写是否正确的信息。当返回值是 0,表示文件读写正确,否则,表示文件读写错误。文件打开时,该值会自动置为 0。

2. 清除文件错误标志 clearerr(fp)

一般形式为:

clearerr(fp) ;

fp 是文件指针,函数 clearerr 的功能是将 fp 文件的错误标志和文件结束标志清除。当文件读写出错时,错误标志(非 0 值)一直保留,可以调用 clearerr(fp)函数将其清 0。

【例 12.7】　文件出错函数实例。

```
#include <stdio. h>
void main(void)
```

```
{
   FILE *stream;
   stream = fopen("s.dat","w");
   getc(stream);                          /*从一个只能写的文件读数据,出错! */
   if (ferror(stream))                    /* 测试错误 */
   {
      printf("Error reading from s.dat\n");   /* 如果出错,显示错误提示 */
      clearerr(stream);                       /*清除错误标志. */
   }
   fclose(stream);
}
```

12.7 本章小结

C系统把文件当作一个“流”,按字节进行处理;C文件按编码方式分为二进制文件和文本文件。

C语言中,用文件指针标识文件,当一个文件被打开时,可取得该文件指针。文件在读写之前必须打开,读写结束必须关闭;文件可按只读、只写、读写、追加四种操作方式打开,同时还必须指定文件的类型是二进制文件还是文本文件;文件可按字节,字符串,数据块为单位读写,文件也可按指定的格式进行读写;文件内部的位置指针可指示当前的读写位置,移动该指针可以对文件实现随机读写。

习 题

一、选择题

1. 若fp是指向某文件的指针,且已读到此文件末尾,则库函数feof(fp)的返回值是_____。

   A) EOF　　B) 0　　C)非零值　　D) NULL

2. 在C语言中,文件的存取方式是以_____为单位。

   A) 记录　　B) 结构　　C) 字符　　D) 字节

3. 在C语言中,文件若按数据的存储形式分类可分为_____。

   A) 字符文件、数字文件　　B) 文本文件、二进制文件

   C) 顺序文件、随机文件　　D) 以上均不对

4. 若要向文件末尾添加新的数据,则应以_____打开文件。

   A) “r”方式　　B)“w”方式

   C)“a”方式　　D)“rb”方式

5. 从键盘输入字符串时,应该_____。

   A)使用单引号　　B)使用双引号

   C)不使用任何符号　　D)以上均可

二、填空题

1. 在磁盘上存储的信息是按照_____形式组织的。

2. 文件的扩展名通常用于_____。

3. 若 fp 已经正确指向一指定文件，则将字符变量 ch 中的字符输出到该文件中，可用语句有_____、_____、_____和_____。

4. 下面程序把从终端读入的文本(用@作为文本结束标志)输出到一个名为 bi.dat 的新文件中。

```
#include <stdio.h>
void main(void)
{
   FILE *fp;char ch;
   if( (fp=fopen (_____) )== NULL)exit(0);
   while( (ch=getchar( ))!='@') fputc (ch,fp);
   fclose(fp);
}
```

三、编写程序题

1. 编程建立一个新文件，文件名通过命令行参数输入，然后，从键盘输入的一串以'#'结尾的字符串，将字符串存到文件中去。

2. 编写程序实现将文件 test2.txt 的内容追加到文件 test1.txt 的尾部(test1.txt、test2.txt 由 1.题的程序生成)。

3. 已知一个班有 45 个人，本学期有两门课程的成绩，从键盘输入每个学生的学号、姓名、成绩(如 200272334,wang jin,94,78)，将其存入文件 stu.rec 中。

4. 输入一个学号在上题形成的文件 stu.rec 中查找，显示结果。

附　　录

1. 常用字符与 ASCII 代码对照表

ASCII值	字符	控制字符	ASCII值	字符	ASCII值	字符	ASCII值	字符	ASCII值	字符	ASCII值	字符	ASCII值	字符	ASCII值	字符
000	(null)	NUL	032	(space)	064	@	096	′	128	Ç	160	à	192	└	224	α
001	☺	SOH	033	!	065	A	097	a	129	ü	161	í	193	┴	225	β
002	●	STX	034	″	066	B	098	b	130	é	162	ó	194	┬	226	Γ
003	♥	ETX	035	#	067	C	099	c	131	â	163	ú	195	├	227	π
004	♦	EOT	036	$	068	D	100	d	132	ä	164	ñ	196	─	228	Σ
005	♣	END	037	%	069	E	101	e	133	à	165	Ñ	197	┼	229	σ
006	♠	ACK	038	&	070	F	102	f	134	å	166	a	198	╞	230	μ
007	(beep)	BEL	039	′	071	G	103	g	135	ç	167	o	199	╟	231	τ
008	■	BS	040	(	072	H	104	h	136	ê	168	¿	200	╚	232	Φ
009	(tab)	HT	041	)	073	I	105	i	137	ë	169	⌐	201	╔	233	θ
010	(line feed)	LF	042	*	074	J	106	j	138	è	170	¬	202	╩	234	Ω
011	(home)	VT	043	+	075	K	107	k	139	ï	171	1/2	203	╦	235	δ
012	(form feed)	FF	044	,	076	L	108	l	140	î	172	1/4	204	╠	236	∞
013	(carriage return)	CR	045	–	077	M	109	m	141	ì	173	¡	205	═	237	∮
014	♫	SO	046	。	078	N	110	n	142	Ä	174	≪	206	╬	238	∈
015	☼	SI	047	/	079	O	111	o	143	Å	175	≫	207	╧	239	∩
016	►	DLE	048	0	080	P	112	p	144	É	176	▒	208	╨	240	≡
017	◄	DC1	049	1	081	Q	113	q	145	æ	177	▓	209	╤	241	±
018	↕	DC2	050	2	082	R	114	r	146	Æ	178	▓	210	╥	242	≥
019	!!	DC3	051	3	083	S	115	s	147	ô	179	│	211	╙	243	≤
020	¶	DC4	052	4	084	T	116	t	148	ö	180	┤	212	╘	244	⌠
021	§	NAK	053	5	085	U	117	u	149	ò	181	╡	213	╒	245	⌡
022	▬	SYN	054	6	086	V	118	v	150	û	182	╢	214	╓	246	÷
023	↨	ETB	055	7	087	W	119	w	151	ù	183	╖	215	╫	247	≈
024	↑	CAN	056	8	088	X	120	x	152	ÿ	184	╕	216	╪	248	°
025	↓	EM	057	9	089	Y	121	y	153	ö	185	╣	217	┘	249	∙
026	→	SUB	058	:	090	Z	122	z	154	ü	186	║	218	┌	250	·
027	←	ESC	059	;	091	[	123	{	155	¢	187	╗	219	█	251	√
028	∟	FS	060	<	092	\	124	¦	156	£	188	╝	220	▄	252	∏
029	◆	GS	061	=	094	]	125	}	157	¥	189	╜	221	▌	253	Z
030	▲	RS	062	>	093	^	126	~	158	Pt	190	╛	222	▐	254	■
031	▼	US	063	?	095	–	127	⌂	159	ƒ	191	┐	223	▀	255(blank 'FF')	

注:128~255 是 IBM-PC(长城 0520)上专用的。表中 000~127 是标准的。

2. C 语言运算符的优先级与结合性

优先级	运算符	功　能	适用范围	结合性
15	() [] . —>	整体表达式、参数表 下标 存取成员 通过指针存取的成员	表达式 参数表 数组 结构/联合	→
14	! ~ ++ —— — & * (type) sizeof	逻辑非 按位求反 加 1 减 1 取负 取地址 取内容 强制类型转换 计算占用内存长度	逻辑运算 位运算 自增 自减 算术运算 指针 指针 类型转换 变量/数据类型	←
13	* / %	乘 除 整数取模	算术运算	→
12	+ —	加 减		
11	<< >>	位左移 位右移	位运算	→
10	< <= > >=	小于 小于等于 大于 大于等于	关系运算	→
9	== !=	恒等于 不等于		
8	&	按位与	位运算	→
7	^	按位异或		
6	|	按位或		
5	&&	逻辑与	逻辑运算	→
4	||	逻辑或		
3	?:	条件运算	条件	←
2	= op=	运算且赋值 op 可为下列运算符之一：*、/、%、+、—、<<、>>、&、^、|		←
1	,	顺序求值	表达式	→

说明：

1. 表中运算符优先级的序号越大，表示优先级别越高。

2. 结合性表示相同优先级的运算符在运算过程中应当遵循的次序。其中符号“→”表示同优先级运算符的运算次序要自左向右进行；符号“←”表示同优先级运算符的次序要自右向左进行。

3. C库函数

C库函数是由编译程序根据一般用户的需要编制并提供用户使用的一组程序。每一种C编译系统都提供了一批库函数,不同的编译系统所提供的库函数的数目和函数名以及函数功能是不完全相同的。ANSI C标准提出了一批建议提供的标准库函数。它包括了目前多数C编译系统所提供的库函数,但也有一些是某些C编译系统未曾实现的。本书列出ANSI C标准建议提供的部分常用库函数。

一、数学函数

数学函数的原型在math.h中。

数学函数表

函数名称	函数与形参类型	函数功能	返回值
acos	double acos(x) double x;	计算 $\cos^{-1}(x)$的值。 $-1 \leqslant x \leqslant 1$	计算结果
asin	double asin(x) double x;	计算 $\sin^{-1}(x)$的值。 $-1 \leqslant x \leqslant 1$	计算结果
atan	double atan(x) double x;	计算 $\tan^{-1}(x)$的值。	计算结果
atan2	double atan2(x,y) double x,y;	计算 $\tan^{-1}(x/y)$的值。	计算结果
cos	double cos(x) double x;	计算cos(x)的值。 x的单位为弧度	计算结果
cosh	double cosh(x) double x;	计算x的双曲余弦cosh(x)的值。	计算结果
exp	double exp(x) double x;	求 e^x 的值。	计算结果
fabs	double fabs(x) double x;	求x的绝对值。	计算结果
floor	double floor(x) double x;	求不大于x的最大整数。	该整数的双精度实数
fmod	double fmod(x,y) double x,y;	求整除x/y的余数。	返回余数的双精度实数
frexp	double frexp(val,eptr) double val; int * eptr;	把双精度数val分解为数字部分(尾数)和以2为底的指数n,即val=x * 2n,n存放在eptr指向的变量中。	数字部分x 0.5≤x<1
log	double log(x) double x;	求 $\log_e x$ 即lnx。	计算结果

续表

函数名称	函数与形参类型	函数功能	返回值
log10	double log10(x) double x;	求 $\log_{10}x$。	计算结果
modf	double modf(val,iptr) double val; double * iptr;	把双精度数 val 分解为整数部分和小数部分,把整数部分存到 iptr 指向的单元。	val 的小数部分
pow	double pow(x,y) double x,y;	计算 x^y 的值。	计算结果
sin	double sin(x) double x;	计算 sin(x)的值 x 的单位为弧度。	计算结果
sinh	double sinh(x) double x;	计算 x 的双曲正弦函数 sinh(x)的值。	计算结果
sqrt	double sqrt(x) double x;	计算$\sqrt{x}$(x≥0)。	计算结果
tan	double tan(x) double x;	计算 tan(x)的值。 x 单位为弧度	计算结果
tanh	double tanh(x) double x;	计算 x 的双曲正切函数 tanh(x)的值。	计算结果

二、字符函数

字符函数的原型在 ctype.h 中。

字符函数表

函数名称	函数与形参类型	函数功能	返回值
isalnum	int isalnum(ch) int ch;	检查 ch 是否字母或数字。	是字母或数字返回 1;否则返回 0。
isalpha	int isalpha(ch) int ch;	检查 ch 是否字母。	是字母,返回 1;否则返回 0。
iscntrl	int iscntrl(ch) int ch;	检查 ch 是否控制字符(其 ASCII 码在 0 和 0x1F 之间)。	是控制字符,返回 1;否则返回 0。
isdigit	int isdigit(ch) int ch;	检查 ch 是否数字(0～9)。	是数字返回 1;否则返回 0。
isgraph	int isgrasph(ch) int ch;	检查 ch 是否是可打印字符(其 ASCII 码在 0x21 到 0x7e 之间)不包括空格。	是可打印字符,返回 1;否则返回 0。
islower	int islower(ch) int ch;	检查 ch 是否是小写字母(a～z)。	是小写字母,返回 1;否则返回 0。
isprint	int isprint(ch) int ch;	检查 ch 是否可打印字符(不包括空格),其 ASCII 码值在 0x20 到 0x7e 之间。	是可打印字符,返回 1;否则返回 0。
ispunct	int ispunct(ch) int ch;	检查 ch 是否标点字符(不包括空格),即除字母、数字和空格以外的所有可打印字符。	是标点,返回 1;否则返回 0。
isspace	int isspace(ch) int ch;	检查 ch 是否空格、跳格符(制表符)或换行符。	是,返回 1;否则返回 0。

续表

函数名称	函数与形参类型	函数功能	返回值
isupper	int isupper(ch) int ch;	检查 ch 是否是大写字母(A～Z)	是大写字母,返回 1;否则返回 0。
isxdigit	int isxdigit(ch) int ch;	检查 ch 是否一个 16 进制数字(即 0～9,或 A～F,a～f)。	是,返回 1;否则返回 0。
tolower	int tolower(ch) int ch;	将 ch 字符转换为小写字母。	返回 ch 对应的小写字母。
toupper	int toupper(ch) int ch;	将 ch 字符转换为大写字母。	返回 ch 对应的大写字母。

三、字符串函数

字符串函数的原型在 string.h 中。

字符串函数表

函数名称	函数与形参类型	函数功能	返回值
memchr	void memchr(buf,ch,count) void * buf;char ch; unsigned int count;	在 buf 的前 count 个字符里搜索字符 ch 首次出现的位置。	返回指向 buf 中 ch 第一次出现的位置指针;若没有找到 ch,返回 NULL。
memcmp	int memcmp(buf1,buf2,count) void * buf1, * buf2; unsigned int count;	按字典顺序比较由 buf1 和 buf2 指向的数组的前 count 个字符。	buf1＜buf2,为负数 buf1＝buf2,返回 0 buf1＞buf2,为正数
memcpy	void * memcpy(to,from,count) void * to, * from; unsigned int count;	将 from 指向的数组中的前 count 个字符拷贝到 to 指向的数组中。from 和 to 指向的数组不允许重叠。	返回指向 to 的指针
mem-move	void * memmove(to,from,count) void * to, * from; unsigned int count;	将 from 指向的数组中的前 count 个字符拷贝到 to 指向的数组中。from 和 to 指向的数组可以允许重叠。	返回指向 to 的指针
memset	void * memset(buf,ch,count) void * buf;char ch; unsigned int count;	将字符 ch 拷贝到 buf 所指向的数组的前 count 个字符中。	返回 buf。
strcat	char * strcat(str1,str2) char * str1, * ctr2;	把字符串 str2 接到 str1 后面,取消原来 str1 最后面的串结束符'\0'。	返回 str1。
strchr	char * strchr(str,ch) char * str; int ch;	找出 str 指向的字符串中第一次出现字符 ch 的位置。	返回指向该位置的指针,如找不到,则应返回 NULL。
strcmp	int strcmp(str1,str2) char * str1, * str2;	比较字符串 str1 和 str2。	str1＜str2,为负数 str1＝str2,返回 0 str1＞str2,为正数
strcpy	char * strcpy(str1,str2) char * str1, * ctr2;	把 str2 指向的字符串拷贝到 str1 中去。	返回 str1。

续表

函数名称	函数与形参类型	函数功能	返回值
strlen	unsigned int strlen(str) char * str;	统计字符串 str 中字符的个数(不包括终止符'\0')。	返回字符个数。
strset	char * strset(but,ch) char * buf;char ch;	将 buf 所指向字符串中的全部字符都变为字符 ch。	返回 buf。
strstr	char * strstr(str1,str2) char * str1, * str2;	寻找 str2 指向的字符串在 str1 指向的字符串中首次出现的位置。	返回 str2 指向的子串首次出现的地址。否则返回 NULL。

四、输入输出函数

输入输出函数的原型在 stdio.h 中。

输入输出函数表

函数名称	函数与形参类型	函数功能	返回值
clearerr	void clearerr(fp) FILE * fp;	清除文件指针错误。	无。
close	int close(fp) int fp;	关闭文件(非 ANSI 标准)。	关闭成功返回 0,不成功,返回-1。
creat	int creat(filename,mode) char * filename; int mode;	以 mode 所指定的方式建立文件(非 ANSI 标准)。	成功则返回正数,否则返回-1。
eof	int eof(fd) int fd;	判断文件(非 ANSI 标准)是否结束。	遇文结束,返回 1;否则返回 0。
fclose	int fclose(fp) FILE * fp;	关闭 fp 所指的文件,释放文件缓冲区。	关闭成功返回 0;否则返回非 0。
feof	int feof(fp) FILE * fp;	检查文件是否结束。	遇文件结束符返回非 0,否则返回 0。
ferror	int ferror(fp) FILE * fp;	测试 fp 所指的文件是否有错误。	无错返回 0;否则返回非 0。
fflush	int fflush(fp) FILE * fp;	将 fp 所指的文件的全部控制信息和数据存盘。	存盘正确返回 0;否则返回非 0。
fgetc	int fgetc(fp) FILE * fp;	从 fp 指向的文件中取得下一个字符。	返回得到的字符。若出错返回 EOF。
fgets	char * fgets(buf,n,fp) char * buf;int n; FILE * fp;	从 fp 指向的文件读取一个长度为(n-1)的字符串,存入起始地址为 buf 的空间。	返回地址 buf,若遇文件结束或出错,则返回 EOF。
fopen	FILE * fopen(filename,mode) char * filename, * mode;	以 mode 指定的方式打开名为 filename 的文件。	成功,返回一个文件指针;否则返回 0。
fprintf	int fprintf(fp,format,args,…) FILE * fp; char * format;	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中。	实际输出的字符数。
fputc	int fputc(ch,fp) char ch; FILE * fp;	将字符 ch 输出到 fp 指向的文件中。	成功,则返回该字符,否则返回 EOF。

续表

函数名称	函数与形参类型	函数功能	返回值
fputs	int fputs(str,fp) char str; FILE * fp;	将 str 指向的字符串输出到 fp 所指定的文件。	成功返回 0,若出错返回 EOF。
fread	int fread(pt,size,n,fp) char * pt; unsigned size; unsigned n; FILE * fp;	从 fp 所指定持文件中读取长度为 size 的 n 个数据项,存到 pt 所指向的内存区。	返回所读的数据项个数,如遇文件结束或出错,返回 0。
fscanf	int fscanf(fp,format,args,…) FILE * fp; char format;	从 fp 指定的文件中按给定的 format 格式将读入的数据送到 args 所指向的内存变量中(args 是指针)。	已输入的数据个数。
fseek	int fseek(fp,offset,base) FILE * fp; long offset; int base;	将 fp 所指向的文件的位置指针移到 base 所指出的位置为基准、以 offset 为位移量的位置。	返回当前位置,否则,返回-1。
ftell	long ftell(fp) FILE * fp;	返回 fp 所指向的文件中的读写位置。	返回文件中的读写位置,否则返回 0。
fwrite	int fwrite(ptr,size,n,fp) char * ptr; FILE * fp; unsigned size,n;	把 ptr 所指向的 n * size 个字节输出到 fp 所指向的文件中。	写到 fp 文件中的数据项的个数。
getc	int getc(fp) FILE * fp	从 fp 指向的文件中读入下一个字符。	返回读入的字符;若文件结束或出错返回 EOF。
getchar	int getchar(void)	从标准输入设备读取下一个字符。	返回字符。若文件结束或出错返回-1。
gets	char * gets(str) char * str;	从标准输入设备读取字符串存入 str 指向的数组。	成功返回指针 str,否则返回 NULL。
open	int open(filename,mode) char * filename; int mode;	以 mode 指定的方式打开已存在的名为 filename 的文件(非 ANSI 标准)。	返回文件号(正数);如文件打开失败,返回-1。
printf	int printf(format,args,…) char * format;	在 format 指定的字符串的控制下,将输出列表 args 的值输出到标准输出设备。	输出字符的个数。若出错,则返回负数。
putc	int putc(ch,fp) int ch; FILE * fp;	把一个字符 ch 输出到 fp 所指的文件中。	输出的字符 ch。若出错,返回 EOF。
prtchar	int putchar(ch) char ch;	把字符 ch 输出到标准输出设备。	输出字符 ch,若出错,则返回 EOF。
puts	int puts(str) char * str;	把 str 指向的字符串输出到标准输出设备,将'/0'转换为回车换行。	返回换行符,若失败,返回 EOF。

续表

函数名称	函数与形参类型	函数功能	返回值
putw	int putw(I,fp) int I; FILE * fp;	将一个整数 I(即一个字)写到 fp 所指的文件(非 ANSI 标准)中。	返回输出的整数;若出错,返回 EOF。
read	int read(fd,buf,count) int fd; char * buf; unsigned int count;	从文件号 fd 所指示的文件(非 ANSI 标准)中读 count 个字节到由 buf 指示的缓冲区中。	返回真正读入的字节个数,如遇文件结束返回 0,出错返回 −1。
remove	int remove(fname) char * fname;	删除以 fname 为文件名的文件。	成功返回 0;出错返回 −1。
rename	int rename(oname,nname) char * oname, * nname;	把 oname 所指的文件名改为由 nname 所指的文件名。	成功返回 0;出错返回 −1。
rewind	void rewind(fp) FILE * fp;	将 fp 指定的文件指针置于文件头,并清除文件结束标志和错误标志。	无。
scanf	int scanf(format,args,…) char * format;	从标准输入设备按 format 指示的格式字符串规定的格式,输入数据给 args 所指示的单元。args 为指针。	读入并赋给 args 数据个数。遇文件结束返回 EOF;若出错返回 0。
write	int write(fd,buf,count) int fd; char * buf; unsigned count;	从 buf 指示的缓冲区输出 count 个字符到 fd 所指的文件(非 ANSI 标准)中。	返回实际输出的字节数,如出错返回 −1。

五、动态存储分配函数

动态存储分配函数的原型在 stdlib.h 中。

动态存储分配函数表

函数名称	函数与形参类型	函数功能	返回值
calloc	void * calloc(n,size) unsigned n; unsigned size;	分配 n 个数据项的内存连续空间,每个数据项的大小为 size。	分配内存单元的起始地址。如不成功,返回 0。
free	void free(p) void * p;	释放 p 所指的内存区。	无。
malloc	void * malloc(size) unsigned size;	分配 size 字节的内存区。	所分配的内存区地址,如内存不够,返回 0。
realloc	void * realloc(p,size) void * p; unsigned size;	将 p 所指的已分配的内存区的大小改为 size,size 可以比原来分配的空间大或小。	返回指向该内存区的指针。若重新分配失败,返回 NULL。

六、其他函数

“其他函数”是C语言的标准库函数，由于不便归入某一类，所以单独列出。函数的原型在 stdlib.h 中。

其他函数表

函数名称	函数与形参类型	函数功能	返回值
abs	int abs(num) int num;	计算整数 num 的绝对值。	返回计算结果。
atof	double atof(str) char * str;	将 str 指向的字符串转换为一个 double 型的值。	返回双精度计算结果。
atoi	int atoi(str) char * str;	将 str 指向的字符串转换为一个 int 型的整数。	返回转换结果。
atol	long atol(str) char * str;	将 str 所指向的字符串转换为一个 long 型的整数。	返回转换结果。
exit	void exit(status) int status;	终止程序运行。将 status 的值返回调用的过程。	无。
itoa	char * itoa(n,str,radix) int n,radix; char * str;	将整数 n 的值按照 radix 进制转换为等价的字符串，并将结果存入 str 指向的字符串中。	返回一个指向 str 的指针。
labs	long labs(num) long num;	计算长整数 num 的绝对值。	返回计算结果。
ltoa	char * ltoa(n,str,radix) long int n; int radix; char * str;	将长整数 n 的值按照 radix 进制转换为等价的字符串，并将结果存入 str 指向的字符串中。	返回一个指向 str 的指针。
rand	int rand()	产生 0 到 RAND－MAX 之间的伪随机数。RAND－MAX 在头文件中定义。	返回一个伪随机(整)数
random	int random(num) int num;	产生 0 到 num 之间的随机数。	返回一个随机(整)数。
random-ize	void randomize()	初始化随机函数。 使用时要求包含头文件 time.h。	无。
strtod	double strtod(start,end) char * start; char * * end;	将 start 指向的数字字符串转换成 double，直到出现不能转换为浮点数的字符为止，剩余的字符串赋给指针 end。 * HUGE－VAL 是 Turbo C 在头文件 math.h 中定义的数学函数溢出标志值。	返回转换结果。若未转换则返回 0。若转换出错，返回 HU－GE－VAL 表示上溢，或返回－HUGE－VAL 表示下溢。
strtol	long int strtol(start,end,radix) char * start; char * * end; int radix;	将 start 指向的数字字符串转换成 long，直到出现不能转换为长整型数的字符为止，剩余的字符串赋给指针 end。转换时，数字的进制由 radix 确定。 * LONG－MAX 是 Turbo C 在头文件 limits.h 中定义的 long 型可表示的最大值。	返回转换结果。若未转换则返回 0。若转换出错，返回 LONG－VAL 表示上溢，或返回－LONG－VAL 表示下溢。
system	int system(str) char * str;	将 str 所指向的字符串作为命令传递给 DOS 的命令处理器。	返回所执行命令的退出状态。

4. 常见错误信息表

常见的出错、警告提示信息(包括编译、连接和运行时的错误提示信息)有:

Ambiguous operators need parentheses	不明确的运算,需要用括号括起
Ambiguous symbol'xxx'	不明确的符号 xxx
Argument list syntax error	参数表语法错误
Array bounds missing	丢失数组界限符
Array size too large	数组长度太大
Bad character in parameters	参数中有不适当的字符
Bad file name format in include directive	包含命令中文件名格式不正确
Bad ifdef directive syntax	编译预处理 ifdef 有语法错
Bad ifndef directive syntax	编译预处理 ifndef 有语法错
Bad undef directive syntax	编译预处理 undef 有语法错
Bit field too large	位段太长
Call of non—function	调用未定义的函数
Call to function with no prototype	调用函数时没有函数的声明
Cannot modify a const object	不允许修改常量对象
Case outside of switch	case 出现在 switch 语句之外
Case statement missing	漏掉了 case 语句
Case syntax error	case 语法错误
Code has no effect	代码不可达(不可能执行到)
Compound statement missing}	复合语句漏掉“}”
Conflicting type modifiers	类型说明符不一致
Constant expression required	需要常量表达式
Constant out of range in comparison	在比较中常量超出范围
Conversion may lose significant digits	转换时会丢失有意义的数字
Conversion of near pointer not allowed	不允许转换近指针
Could not find file'xxx'	找不到 xxx 文件
Declaration missing;	说明缺少“;”
Declaration syntax error	说明中出现语法错误
Default outside of switch	default 出现在 switch 语句之外
Define directive needs an identifier	定义编译预处理需要标识符
Division by zero	用零作除数
Do statement must have while	do while 语句中缺少 while 部分
Enum syntax error	枚举类型语法错误
Enumeration constant syntax error	枚举常数语法错误
Error directive:xxx	错误的编译预处理命令 xxx
Error writing output file	写输出文件错误
Expression syntax error	表达式语法错误
Extra parameter in call	调用时出现多余参数
File name too long	文件名太长
Function call missing)	函数调用缺少右括号
Function definition out of place	函数定义位置错误
Function should return a value	函数必须返回一个值

Goto statement missing label	goto 语句没有标号
Hexadecimal or octal constant too large	十六进制或八进制常数太大
Illegal character ‘x’	非法字符 x
Illegal initialization	非法的初始化
Illegal octal digit	非法的八进制数字
Illegal pointer subtraction	非法的指针相减
Illegal structure operation	非法的结构操作
Illegal use of floating point	非法的浮点运算
Illegal use of pointer	指针使用非法
Improper use of a typedef symbol	类型定义符使用不恰当
In - line assembly not allowed	不允许使用嵌入汇编
Incompatible storage class	存贮类别不相容
Incompatible type conversion	不相容的类型转换
Incorrect number format	错误的数据格式
Incorrect use of default	default 使用不正确
Invalid indirection	无效的间接运算
Invalid pointer addition	指针相加无效
Irreducible expression tree	无法执行的表达式运算
Lvalue required	必须用变量左值
Macro argument syntax error	宏参数语法错误
Macro expansion too long	宏展开以后太长
Mismatched number of parameters in definition	定义中参数个数不匹配
Misplaced break	此处不应出现 break 语句
Misplaced continue	此处不应出现 continue 语句
Misplaced decimal point	此处不应出现小数点
Misplaced elif directive	此处不应出现编译预处理 elif
Misplaced else	此处不应出现 else
Misplaced else directive	此处不应出现 else
Misplaced endif directive	此处不应出现编译预处理 endif
Must be addressable	必须是可以编址的
Must take address of memory location	必须得到定位的内存地址
No declaration for function ‘xxx’	没有函数 xxx 的声明
No stack	缺少堆栈
No type information	没有类型信息
Non - portable pointer assignment	不可移动的指针(地址常数)赋值
Non - portable pointer comparision	不可移动的指针(地址常数)比较
Non - portable pointer conversion	不可移动的指针(地址常数)转换
Not a valid expression format type	不合法的表达式格式
Not an allowed type	不允许使用的类型
Numeric constant too large	数值常数太大
Out of memory	内存不够用
Parameter ‘xxx’ is never used	参数 xxx 没有用到
Pointer required on left side of ->	->符号的左边必须是指针
Possible use of ‘xxx’ before definition	在定义之前就使用了 xxx(警告)
Possibly incorrect assignment	赋值可能不正确
Redeclaration of ‘xxx’	重复定义了 xxx

Redefinition of ‘xxx’ is not identical	xxx 的两次定义不一致
Register allocation failure	寄存器定址失败
Repeat count needs an lvalue	重复计数需要逻辑值
Size of structure or array not known	结构体或数组的大小不确定
Statement missing;	语句后缺少“;”
Structure or union syntax error	结构体或共用体语法错误
Structure size too large	结构体的大小太大
Subscripting missing]	下标缺少右方括号
Superfluous & with function or array	函数或数组中有多余的“&”
Suspicious pointer conversion	可疑的指针转换
Symbol limit exceeded	符号超限
Too few parameters in call	函数调用时的实参少于函数的参数
Too many default cases	default 太多(switch 中只有一个)
Too many error or warning messages	错误或警告信息太多
Too many types in declaration	声明中类型太多
Too much auto memory in function	函数中用到的局部存贮太多
Too much global data defined in file	文件中全局数据太多
Two consecutive dots	两个连续的句点
Type mismatch in parameter ‘xxx’	参数 xxx 类型不匹配
Type mismatch in redeclaration of ‘xxx’	xxx 重定义时的类型不匹配
Unable to create output file ‘xxx’	无法建立输出文件 xxx
Unable to open include file ‘xxx’	无法打开被包含的文件 xxx
Unable to open input file - ‘xxx’	无法打开输入文件 xxx
Undefined label ‘xxx’	没有定义标号 xxx
Undefined structure ‘xxx’	没有定义结构体 xxx
Undefined symbol ‘xxx’	没有定义符号 xxx
Unexpected end of file in comment started on line ‘xxx’	从 xxx 行开始的注解尚未结束,文件不能结束
Unexpected end of file in conditional started on line ‘xxx’	从 xxx 行开始的条件语句尚未结束,文件不能结束
Unknown assemble instruction	未知的汇编指令
Unknown operation	未知操作
Unknown preprocesser directive ‘xxx’	不认识的预处理命令 xxx
Unreachable code	无路可达的代码
Unterminated string or character constant	字符串缺少引号
User break	用户强行中断了程序
Void functions may not return a value	void 类型的函数不应有返回值
Wrong number of arguments	调用函数时参数数目错
‘xxx’ not an argument	xxx 不是参数
‘xxx’ not part of structure	xxx 不是结构体的一部分
‘xxx’ statement missing (	xxx 语句缺少左括号
‘xxx’ statement missing)	xxx 语句缺少右括号
‘xxx’ statement missing;	xxx 语句缺少分号
‘xxx’ declared but never used	说明了 xxx,但没有使用
‘xxx’ is assigned a value which is never used	给 xxx 赋了值,但未用过
Zero length structure	结构体的长度为零

参 考 文 献

[1] 谭浩强.C程序设计教程. 北京:清华大学出版社,2007.
[2] 吴国凤.C/C++程序设计(第2版). 北京:高等教育出版社,2008.
[3] 黄维道.C语言程序设计. 北京:清华大学出版社,2003.
[4] 张基温.C程序设计案例教程. 北京:清华大学出版社,2004.
[5] 杨旭.C语言程序设计案例教程. 北京:人民邮电出版社,2005.
[6] 何钦铭.C语言程序设计. 北京:高等教育出版社,2008.

责任编辑◆汤礼广
封面设计◆诚邦视觉设计
ZYZY8888@SINA.COM

ISBN 978-7-5650-0871-9
9 787565 008719 >
定价：54.00 元